UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN:
SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

BIBLIOTECA PARA PORTABILIDAD DE PERFILES EN ENTORNOS DE SSO (SINGLE SIGN-ON)

Autor: Alberto Bermejo Gil
Tutor: Andrés Marín López
Co–director: Daniel Diaz Sanchez

13 de marzo de 2011

Agradecimientos

Escribir éstas líneas supone una gran alegría para mí, porque por fin se pone punto y final a una etapa de mi vida, que me ha dado buenos y malos momentos y que me ha dado la posibilidad de hacer amigos que sé que son para toda la vida.

Muchísimas gracias por supuesto a mi tutor Andrés, por la ayuda que me ha prestado, y la dedicación y paciencia que ha tenido conmigo. También gracias a mi Co–director Daniel, por tener la idea original de este proyecto.

Por supuesto no puedo dejar sin mencionar a todos Mis amigos, Muchísimas gracias a todos por conseguir que este camino haya sido más fácil, porque con una sonrisa todo es mucho más sencillo.

Y en último lugar quiero dar las gracias a mi familia, a mis hermanos, gracias por estar conmigo y hacerme la vida más feliz, y en especial a mis padres, que gracias a su esfuerzo han hecho todo esto posible. Gracias por confiar en mí, por apoyarme, por nunca perder la esperanza, por enseñarme todo lo que sé.

Sin vosotros esto no hubiera ocurrido, GRACIAS.

Resumen

Actualmente cuando se quiere acceder a algún servicio web al registrarse hay que introducir un perfil. En la mayoría de las ocasiones se utiliza este mismo perfil para distintas webs. Este perfil, es un conjunto de atributos que constituye la identidad digital. Para hacer este proceso más cómodo han aparecido tecnologías de Single Sign-On, que permiten poder utilizar el mismo identificador, el mismo perfil y los datos almacenados en un servidor para no tener que introducirlos cada vez que el usuario se tenga que registrar en alguna web o autentificarse en algún portal.

Estas tecnologías SSO como XZID, SAML, OpenID.., están implementadas en una gran cantidad de portales Web incrementando la experiencia del usuario. En OpenID simplemente hay que introducir el identificador OpenID y luego autenticarse contra el propio servidor OpenID. Con ello la página web puede obtener los datos de un perfil sin tener que estar replicándolos de nuevo, por lo que evita el tener que recordar los distintos usuarios y contraseñas de todas las Webs en las cuales se está registrado; con un único identificador ya basta.

El problema surge cuando un usuario quiere modificar su perfil de identidad digital, trasladarlo a otro proveedor o simplemente exportarlo. Este perfil reside en el proveedor de identidad. Por tanto tiene que ir al propio servidor a modificarlo o exportarlo. Por otro lado, cuando el usuario quiere cambiar de proveedor de identidad, tiene que copiar manualmente los atributos.

La solución que se plantea en este proyecto es crear una biblioteca que interconecte portales Web, aplicaciones y proveedores de identidad de forma que el perfil del usuario pueda estar en distintos dispositivos y exportarse e importarse sin problema. Para ello las distintas aplicaciones podrán modificar los distintos perfiles de usuario y utilizar la biblioteca para modificarlo, replicarlo o eliminarlo con un solo clic, y así poder tener actualizado el perfil, también la biblioteca posibilita la creación de nuevos perfiles que se adapten al usuario dependiendo de sus necesidades, como pueden ser el uso de distintos dispositivos o redes. Además que también se permita crear, borrar y modificar sus propios perfiles independientemente del dispositivo que utilice, la red que utilice o las condiciones que sean.

Para todo esto, en este proyecto se va a crear una biblioteca que será capaz de ofrecer a los perfiles de usuarios localizados en diferentes proveedores

de identidad, la posibilidad de interactuar con estos datos agregando, modificando, eliminando, exportando e importando su contenido, manteniendo la coherencia de dichos perfiles y el control sobre los datos personales.

Abstract

Currently when you access a Web service to register you must enter a profile. In most cases using this same profile for different sites. This profile is a set of attributes that constitutes the digital identity. To make this process more convenient technologies have appeared Single Sign-On, which allows to use the same identifier, the same profile and data stored on a server to avoid having to enter it each time the user from having to log into any website or log on to a portal.

These technologies SSO are XZID , SAML, OpenID .., are implemented in a large number of Web portals to increase the user experience. In OpenID simply enter the OpenID identifier and then authenticate against in the OpenID server. This web page can obtain data from a profile without having to be replicated again, thus avoiding having to remember different usernames and passwords of all the websites in which they are registered, with a unique identifier Enough .

The problem arises when a user wants to modify their digital identity profile, move to another provider or export this. This page is in the identity provider. Therefore, it must go to the server itself to modify or export this. On the other hand, when the user wants to change the identity provider, you must manually copy the attributes.

The solution proposed in this project is to create an interface library Web portals, applications and identity providers so that the user profile may be on different devices and exported and imported without problem. For different applications it may modify various user profiles and use the library for modify, replicate or remove with one click, so you can have updated the profile, the library also enables the creation of new profiles that fit the user depending their needs, such as the use of different devices or networks. In addition also allow to create, delete and modify their own profiles regardless of device used, the network that uses or conditions are.

For all this, the function of this project is to create a library that will be able to offer user profiles located in differents identity providers, the ability to interact with these data for adding, modifying, deleting, exporting and importing its contents, keeping consistency of these profiles and control over personal data.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El presente documento recoge la explicación y documentación del proyecto biblioteca para portabilidad de perfiles en entornos SSO. En él se da una detallada y completa documentación sobre el desarrollo del diseño e implementación de la biblioteca, así como su funcionalidad. También se explica la utilización del servidor elegido como proveedor de identidad, así como las múltiples pruebas realizadas y la creación de un escenario especifico para poder ver su funcionamiento mediante una aplicación cliente y así poder comprobar la funcionalidad de la biblioteca, donde se mostraran ejemplos gráficos y capturas de pantalla.

1.1. Motivación del proyecto

La motivación de este proyecto se apoya en una situación cotidiana que todo el mundo ha sufrido alguna vez, que consiste en la necesidad de introducir un mismo perfil, usuario y contraseña en todas las aplicaciones web. Es algo muy tedioso el tener que estar introduciendo cada vez que te registras un perfil y en muchos casos se tienen distintos usuarios y contraseñas para diferentes sitios. Este proceso de registro es necesario en la mayoría de las aplicaciones.

Los protocolos de SSO permiten a los usuarios asociar un identificador a un perfil y disponer de autenticación centralizada en su proveedor de identidad. De esta manera, se evita el proceso de registro, únicamente se introduce el identificador, se autentica al usuario y el servicio recoge el perfil del proveedor.

El problema que trata el PFC aparece cuando se desea exportar su perfil para posteriormente importarlo en otro proveedor, o bien editar un perfil en varios proveedores simultáneamente.

También se facilita la interconexión entre distintos proveedores, lo cual hace que no se tenga que estar realizando operaciones similares en distintos sitios y con ello se logre agrupar distintos servicios para que trabajen en común y no tener que estar repitiendo tareas para varios sitios u aplicaciones.

En definitiva, SSO alivia en gran medida el problema de la gestión de identidad pero en su entorno multiproveedor, en el que los perfiles están repartidos en varios proveedores de identidad, gestionar el perfil de usuario, copiarlo o moverlo de un proveedor a otro, sigue siendo una tarea manual.

Al final esto se traduce en una gran cantidad de ventajas, lo que hace pensar que este proyecto puede llegar a ser muy útil y el cual nos puede ahorrar mucho tiempo y esfuerzo gastado inútilmente.

1.2. Objetivos

El objetivo principal de este proyecto es desarrollar una biblioteca que interactué con proveedores de identidad para poder actualizar, copiar e importar perfiles de usuario.

Esta biblioteca utilizara entornos de seguridad SSO, y la cual será capaz de interactuar con un proveedor de identidad para poder actualizar y modificar datos de perfiles. Y además facilitara la integración de distintas aplicaciones, para lo que se tiene que implementar un lenguaje común para que las propias aplicaciones puedan comunicarse entre sí.

También esta biblioteca tiene que tener en cuenta la posibilidad de que se pueda implementar su uso independientemente de dispositivos, en distintas redes y con múltiples tecnologías SSO, para así lograr una aplicación totalmente independiente del entorno en el que nos encontremos.

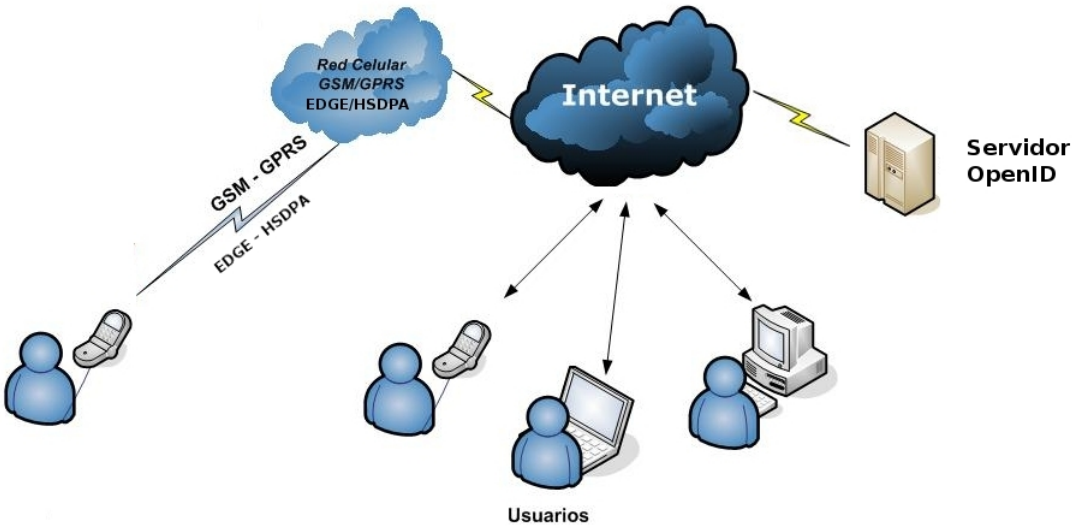

Figura 1.1: Arquitectura de interconexión

1.3. Contenido de la memoria

La memoria consta de ocho capítulos junto con seis apéndices y la bibliografía. En ella se explica el diseño, implementación y funcionalidad del proyecto fin de carrera realizado. Los capítulos son:

- El capítulo 2, contiene el "Estado de la técnica". Dentro de él se muestran las tecnologías que se han consultado e implementado, primera mente se hace una visión del entorno de seguridad en el cual se va a trabajar, hablando de Single Sign-On, SAML, SSL y OpenID, Se profundizara un poco más en OpenID [7] para mostrar ejemplos de distintos tipos de servidores OpenID más usuales actualmente, señalando de cada uno de ellos, sus características más relevantes, lo que permite seleccionar uno de entre todos ellos. Después se dará paso a las herramientas de desarrollo utilizadas como son APML (Attention Profiling Mark-up Language) [3] y JAVA.

- El capítulo 3, describe el diseño y descripción de la biblioteca y como se conecta con el proveedor de identidad, así como el formato de los perfiles que se va a utilizar y el diseño de la aplicación grafica.

- El capítulo 4, habla de las propias funcionalidades implementadas en la biblioteca y describe de forma detallada el funcionamiento, también

se indica como se implementa la aplicación grafica y de como se pone y configura el servidor de identidad digital.

- El capítulo 5, versa sobre las pruebas realizas sobre la propia biblioteca mientras esta se implementaba y después de la implementación con la ayuda de una aplicación gráfica.
- El capítulo 6, se puede ver el presupuesto por el cual es posible la realización de este proyecto.
- El capítulo 7, relata la historia del proyecto en el cual se relata fase a fase la evolución del proyecto, también se habla de los problemas que han ido ocurriendo mientras la realización del mismo.
- El capítulo 8, son las conclusiones finales del proyecto y los trabajos futuros que se pueden llegar a realizar.
- El Apéndice A, es una guía de instalación del servidor WSO2.
- El Apéndice B, se basa en la explicación de como configurar inicialmente el entorno para poder utilizar WSO2 con nuestro propio dominio.
- El Apéndice C, muestra como poder crear un certificado SSL propio.
- El Apéndice D, es un manual para usuarios del servidor.
- El Apéndice E, es una guía del administrador la cual ayuda a poder utilizar todas las características que contiene el servidor.
- El Apéndice F, enseña como introducir almacenes de usuarios externos a nuestro servidor, en este caso centrado en la utilización de LDAP.

Capítulo 2

Estado de la técnica

2.1. Introducción

Para el desarrollo de este proyecto se han utilizado varias tecnologías, estándares y diversos conceptos. A lo largo de éste punto se detallaran todos ellos.

En cada apartado se muestra al lector alguna de sus características principales, una justificación de por qué se han utilizado en el proyecto, y unas referencias bibliográficas para que pueda ampliar la información aquí recogida.

2.2. Identidad digital

La identidad digital es la representación digital de un conjunto de afirmaciones sobre un sujeto, estas pueden estar compuestas por una colección de atributos basados en información de uno mismo o de credenciales que los demás dicen sobre el usuario.

Existen distintos tipos de identidad digital:

- Federada: Las tareas de identificación y establecimiento de los datos pertinentes a un usuario se llevan a cabo en el entorno de la organización a la que pertenece, mientras que los procedimientos para establecer derechos de acceso a los recursos están bajo el control de la organización que los proporciona. Algunos ejemplos son SAML, OpenID o ID-WSF.

- User-centric o centrada en el usuario: El usuario tiene el control total sobre sus datos y selecciona que información proporcionar, a quien y cuando. Algunos ejemplos son Cardspace y OpenID.

- Corporativa: El perfil y las credenciales del usuario de encuentran dentro de una organización y sobre los cuales el usuario no tiene ningún control. Algunos ejemplos son Google Apps, yahoo y las cuentas de usuario de la UC3M.

Con esta identidad un usuario puede identificarse en redes de diferentes departamentos o incluso empresas. De este modo las empresas comparten información sin compartir tecnologías de directorio, seguridad y autenticación, como requieren otras soluciones (metadirectorio, Single Sign On, etc.). Para su funcionamiento es necesaria la utilización de estándares que definan mecanismos que permiten a las empresas compartir información entre dominios.

2.3. Single Sign–On (SSO)

Single sign–on (SSO) es un procedimiento de autenticación que permite al usuario acceder a varios sistemas con una sola instancia de identificación.

Hay cinco tipos principales de SSO [18], también se les llama sistemas de autenticación reducida.

- Enterprise single sign–on (E–SSO), también llamado legacy single sign–on, funciona para una autenticación primaria, interceptando los requisitos de autenticación presentados por las aplicaciones secundarias para completar los mismos con el usuario y contraseña. Los sistemas E-SSO permiten interactuar con sistemas que pueden deshabilitar la presentación de la pantalla de autentificación.

- Web single sign-on (Web-SSO), también llamado Web access management (Web-AM) trabaja sólo con aplicaciones y recursos accedidos vía web. Los accesos son interceptados con la ayuda de un servidor proxy o de un componente instalado en el servidor web destino. Los usuarios no autenticados que tratan de acceder son redirigidos a un servidor de autenticación y regresan solo después de haber logrado un acceso exitoso. Se utilizan cookies, para reconocer aquellos usuarios que acceden y su estado de autenticación.

- Kerberos es un método popular de externalizar la autenticación de los usuarios. Los usuarios se registran en el servidor Kerberos y reciben un "ticket", luego las aplicaciones-cliente lo presentan para obtener acceso.

- Identidad federada es una nueva manera de concebir este tema, también para aplicaciones Web. Utiliza protocolos basados en estándares para habilitar que las aplicaciones puedan identificar los clientes sin necesidad de autenticación redundante.

- OpenID es un proceso de SSO distribuido y descentralizado donde la identidad se compila en una url que cualquier aplicación o servidor puede verificar.

En este proyecto nos vamos a centrar en OpenID dado que es una de las alternativas más extendidas y simples de SSO actuales, además se va a utilizar también Web-SSO para la autentificación OpenID dentro del servidor y así poder realizar un acceso más seguro.

2.3.1. SAML

SAML (Security Assertion Markup Language) [18] es un estandard de OASIS. Que define el intercambio de datos de autenticación y autorización entre dominios de seguridad usando XML, es decir, entre un proveedor de identidad y un proveedor de servicios.

El problema individual que SAML está tratando de resolver es el Single Sign-On (SSO) en la navegación Web, un problema también dirigido por el más ampliamente utilizado estándar OpenID.

SAML 2.0 [18] describe dos roles de federación: el de proveedor del servicio, que es la entidad que da acceso al usuario a un recurso o aplicación; y el de proveedor de identidad, responsable de la autenticación del usuario. Ambos, proveedor del servicio y proveedor de identidad, intercambian mensajes para permitir la firma única y un único log de acceso. Tal intercambio de mensajes puede ser iniciado por cualquiera de las dos entidades.

Para la firma única, el proveedor de identidad se responsabiliza de crear y enviar al proveedor de servicio una "aserto" (assertion), que contiene la identidad del usuario. El proveedor de servicio, por su parte, se hace cargo de validar el aserto SAML antes de permitirle acceder a la aplicación.

Un aserto SAML es un documento XML que contiene diversos elementos relativos a la identidad del usuario, como la forma en que el usuario ha sido autenticado y, opcionalmente, atributos sobre su identidad. El intercambio de tales mensajes puede producirse por medios diferentes, ya sea en forma HTTP o en servicios Web.

La convergencia de las formas de uso que aporta SAML 2.0 tendrá un efecto fundamental sobre el interés de las empresas por el uso de la federación como medio de compartir información sobre identidades sin restricciones. Simplifica, además, la elección de la tecnología a adoptar y elimina la necesidad de soluciones multiprotocolo confusas, complejas y caras de mantener.

Cómo funciona el servicio SSO en SAML:

El estándar de federación SALM 2.0 permite que un sitio Web dé acceso a un usuario que ya ha sido autenticado en otro dominio.

Un usuario intenta acceder a un sitio Web. Si no ha sido autenticado, el sitio redirecciona su navegador a un servidor de federación local.

El servidor de federación local redirige al usuario a un servidor de federación remoto, el cual se encarga de comprobar su identidad. El usuario proporciona su nombre y contraseña.

El servidor de federación remoto verifica la identidad del usuario en un servidor LDAP (Lightweight Directory Access Protocol). Si las credenciales del usuario son validas, el servidor de federación remoto crea un aserto SAML, la devuelve y el otro hace POST (si se trata de binding) y le devuelve al servidor de federación local.

El servidor de federación local extrae el aserto SAML y crea una cookie de sesión. El navegador de usuario se redirige al sitio Web.

2.3.2. Secure Socket Layer (SSL)

Secure Socket Layer (SSL) [11] [14] [17] [19] [18] es un protocolo criptográfico desarrollado por Netscape Communications Corporation en 1996 para dar seguridad a la transmisión de datos.

Este protocolo permite abrir conexiones seguras a través de una red, como por ejemplo Internet, cifrando los datos intercambiados entre cliente y servidor mediante un algoritmo de cifrado simétrico. Para poder intercambiarse la clave de sesión utilizada entre cliente y servidor, se utiliza un algoritmo de cifrado de clave pública, típicamente RSA. Como algoritmo de hash se utilizan SHA-256, SHA1, MD5, etc.

Para cada transacción de envío de datos se genera una clave de sesión distinta, de manera que aunque la clave sea reventada para una transacción, las futuras transacciones no se verán afectadas porque utilizarán otras claves distintas.

La principal área de aplicación de las conexiones SSL es asegurar el comunicación entre el navegador y el servidor web. Pero SSL también se utiliza frecuentemente para asegurar la comunicación entre servidores.

SSL tiene como objetivos:

- Seguridad: mediante el cifrado de datos se garantiza que terceros no podrán tener acceso a la información cuando es enviada a través de la red.
- Integridad de los datos: la información enviada mediante una conexión puede ser validada en los extremos para comprobar que no ha sido alterada durante el camino.
- Autenticidad: nadie puede hacerse pasar por un sitio porque gracias a los algoritmos de encriptación se comprueba que los datos realmente han llegado al servidor que el cliente espera. La autenticidad permite evitar fraudes y ataques como Phishing o Man in the Middle entre otros.

Certificados digitales

Las conexiones SSL requieren que el servidor disponga de un certificado digital, el cual consiste en un archivo que identifica de modo único tanto a individuos como a servidores. Gracias a este archivo, el servidor puede autentificar-se antes de establecer la sesión SSL.

PKI X.509 [18] es un estándar UIT-T para infraestructuras de claves públicas. X.509 especifica, entre otras cosas, formatos estándar para certificados de claves públicas y un algoritmo de validación de la ruta

de certificación. A partir de este estándar, se han desarrollado nuevos estándares que utilizan certificados X.509 en conexiones de correo seguro, comercio electrónico, etc.

Los certificados X.509 contienen información relacionada con el usuario, la entidad emisora y el certificado en sí mismo, además de la firma digital del emisor. La sintaxis se define empleando el lenguaje ASN.1 (Abstract Syntax Notation One) y los formatos de codificación más comunes son DER (Distinguished Encoding Rules) o PEM (Privacy-enhanced Electronic Mail). DER define un formato de datos en binario, mientras que PEM está codificado en Base64.

Los certificados digitales están generalmente firmados por una autoridad de certificación (CA), la cual es fiable y permite garantizar la validez del certificado.

En caso de haber algún problema al versificar el certificado, si se accede mediante un navegador, aparece un mensaje para que sea el propio usuario quien decida si confía o no en el contenido de la página.

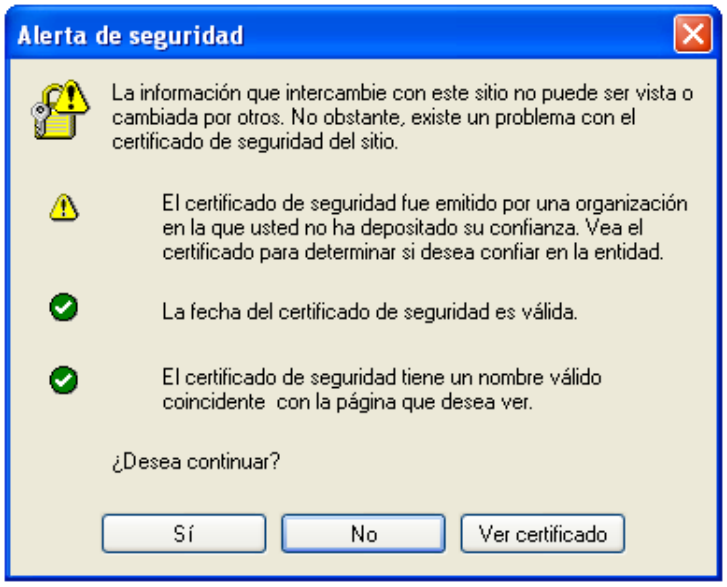

Figura 2.1: Mensaje para preguntar al usuario si desea confiar en el certificado y acceder a la página

El contenido de un certificado digital es el siguiente:

- Clave pública.
- Clave privada
- Información del propietario
- Información del emisor del propietario

Los campos obligatorios de la última versión de un certificado X.509, versión 3, son:

- Versión del estándar X.509 que aplica al certificado.
- Número de serie para distinguir el certificado de los demás, ya que es un identificador único para cada certificado.
- Identificador del algoritmo usado para generar la firma digital del certificado.
- Nombre del emisor del certificado de acuerdo con el estándar X.500.
- Período de validez en el cual el certificado es válido.
- Nombre del sujeto poseedor de la clave pública, el cual es un nombre distinguible (DN, Distinguished Name) de acuerdo con el estándar X.500 o un nombre alternativo como una dirección de correo electrónico o una entrada de DNS.
- Clave pública del sujeto junto con el identificador del algoritmo usado para generar la pareja de claves.
- Algoritmo usado para firmar el certificado.
- Hash de la firma digital.
- Firma digital del certificado.

Además de estos campos, los certificados contienen campos opcionales como el identificador único del emisor y el identificador único del sujeto, que fueron introducidos en la versión 2, y extensiones que pueden ser añadidas al certificado, tales como el uso permitido para la clave, ubicación de la lista de revocación (CRL, Certificate Revocation List), etc. Éstas últimas fueron definidas en la versión 3.

El siguiente certificado muestra un certificado X.509 v3 emitido por Andrés Marín a Alice López. La clave pública es una clave RSA1 de 1024 (módulo y exponente público) y la firma ha sido creada con un resumen digital MD52 de la primera parte del certificado y cifrándola con la clave privada RSA de Andrés Marín.

El certificado de Andrés Marín es un certificado raíz, es decir, un certificado auto–firmado, en el cual el emisor y el sujeto son iguales.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=ES, ST=Madrid, O=IT-UC3M, CN=Andres Marin
        Validity
            Not Before: Oct 10 10:32:40 2008 GMT
            Not After : Oct 10 10:32:40 2009 GMT
        Subject: C=ES, ST=Madrid, O=IT-UC3M, CN=Alice, CN=Lopez
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c8:6d:e9:36:63:57:19:d4:5a:34:ae:38:bf:1b:
                    11:2c:1e:46:52:76:c3:4a:16:17:bd:e6:02:af:9d:
                    3a:0f:1e:47:78:63:37:29:4e:00:69:da:9a:d7:6c:
                    6f:11:42:98:e6:8e:78:72:ad:45:fc:0b:db:6b:a7:
                    12:36:36:c8:10:00:5f:a9:bf:f1:37:0f:34:0e:ba:
                    5d:d7:97:a5:8d:2a:7e:61:ab:98:27:cf:25:3f:ab:
                    54:61:55:5a:82:81:08:5f:83:b3:70:28:df:98:45:
                    32:ef:49:cd:9f:97:7d:3d:73:33:82:81:77:77:23:
                    71:6f:f0:d6:5b:ee:58:53:c5
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:TRUE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                F7:80:25:DE:A4:AD:B5:64:EE:43:15:A7:41:AD:75:30:74:B1:2F:E4
            X509v3 Authority Key Identifier:
                DirName:/C=ES/ST=Madrid/O=IT-UC3M/CN=Andres Marin
                serial:E5:22:D0:96:C2:37:4C:48

    Signature Algorithm: md5WithRSAEncryption
        9e:40:17:12:da:cf:87:28:8a:bd:fa:f1:36:4e:a4:30:dd:68:
        4a:80:43:da:f8:50:16:e7:e9:d1:84:0f:95:29:0e:84:d3:2f:
        60:3b:f4:54:03:01:61:bc:6a:ec:ab:25:61:b7:e7:4e:2c:5e:
        66:e7:f3:9c:2d:e5:2d:6f:23:a1:92:4a:a0:34:3a:f0:28:bb:
        ab:4c:98:41:d8:fd:8e:60:b0:0e:e2:4c:af:d8:d4:fc:72:16:
        c5:a2:ed:79:86:25:c0:0d:f4:6c:ee:b2:bc:9d:6a:a8:1c:2a:
        62:48:2f:3e:3e:8e:60:fc:e7:cc:6a:25:15:c8:eb:2d:e1:09:
        11:f0
```

Figura 2.2: Certificado X.509 v3

Funcionamiento del protocolo

SSL tiene dos fases en su proceso de comunicación: La primera fase utiliza el protocolo SSL handshake. Durante esta fase se establece una comunicación basada en encriptación asimétrica en la cual el cliente y el servidor intercambian una serie de mensajes y negocian los parámetros de la sesión. La segunda fase es en la que se establece la verdadera sesión de comunicación donde las aplicaciones intercambian información.

Las fases del SSL handshake son:

- Client Hello: esta es la fase de presentación del cliente. En ella le indica qué algoritmos de encriptación soporta, le envía un número aleatorio para poder realizar la comunicación segura aún en el caso que el servidor no pueda certificar su validez y pide al servidor que certifique quien es.
- Server Hello: Como respuesta, el servidor también se presenta enviando al cliente su certificado digital encriptado, su llave pública, el algoritmo que usará y otro número aleatorio. El algoritmo que usará será el más potente que soporte tanto el servidor como el cliente.

Tecnologías:

- Aceptación del cliente: El cliente recibe el certificado digital del servidor, lo desencripta usando la llave pública recibida y lo verifica mediante su almacén de certificados. Después realiza verificaciones del certificado por medio de fechas y URL del servidor, entre otros. Finalmente, el cliente genera una llave aleatoria usando la llave pública del servidor y el algoritmo seleccionado y se la envía al servidor cifrada con la propia llave pública del servidor. La llave encriptada es la que se utilizará para la encriptación simétrica durante el intercambio de datos cuando la conexión SSL esté totalmente establecida.
- Verificación: El servidor recibe la llave encriptada, la desencripta con su llave privada y, con el fin de asegurar que nada ha cambiado, ambas partes se envían las llaves. Si coinciden, el handshake concluye y comienza la transacción.

Respecto al fin de la comunicación SSL, cuando el cliente abandona el servidor, se le informa que terminará la sesión segura para luego terminar con SSL.

En el siguiente esquema se muestra el proceso del Handshake del protocolo SSL:

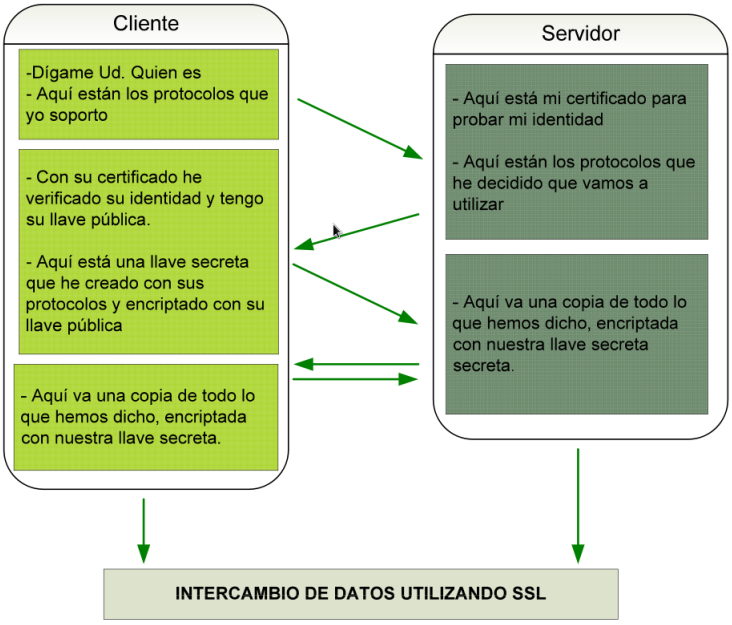

Figura 2.3: Esquema del proceso de Handshake

2.3.3. OpenID

OpenID es una tecnología de código abierto que proporciona un protocolo para que los usuarios de la Web puedan registrar su identidad en un proveedor de identidad y luego usar esa identidad en cualquier lugar de la Web que soporte el protocolo de OpenID [12] [7] [6]. Esto implica que, como usuario, no tiene que seguir creando y recordando nombres de usuarios y contraseñas nuevas en cada sitio que visite. Solo necesitan disponer de un identificador creado en un servidor que verifique OpenID, llamado proveedor de identidad o IdP.

Más de 9 millones de sitios Web, actualmente aceptan OpenID, creciendo mes a mes. Muchas de las principales compañías del mundo, incluyendo AOL, Yahoo, Facebook, MySpace, Twitter, Google Blogger, France Telecom, Telecom Italia han adoptado o anunciado su apoyo a OpenID que utilizan más de 1 billón de usuarios.

A diferencia de otras arquitecturas Single Sign–On, OpenID no especifica el mecanismo de confianza. Por lo tanto, la seguridad de una conexión OpenId depende de la confianza que tenga el cliente OpenID en el proveedor de identidad. Si no existe confianza en el proveedor, la autenticación no será adecuada

para servicios bancarios o transacciones de comercio electrónico, sin embargo el proveedor de identidad puede usar autenticación fuerte pudiendo ser usada para dichos fines.

Funcionamiento

El usuario crea una cuenta en su servidor OpenID. Luego al ir a registrarse o autenticarse en una Web, esta pedirá el usuario y contraseña y habrá una opción mas fácil que es usar una cuenta OpenID, donde se introducirá la URL de la cuenta OpenID ya creada, entonces esta Web hace una petición contra el proveedor OpenID, el cual pedirá la autenticación del usuario y si esta es correcta entonces se producirá la autentificación en dicha web y se podrá hacer como el usuario creado en OpenID.

Ventajas

Ante todo la centralización de la información, no es necesario más altas, más registros, no más correos de confirmación y sobre todo, que una vez autenticado por primera vez contra el servidor, no es necesario hacer lo más veces. Con introducir la Url de la cuenta OpenID es suficiente.

También aporta el minimizar los riesgos de seguridad por contraseña.

Posibilita un registro y entrada a Webs más fácil y rápido.

Inconvenientes

No todas las webs soportan el uso de OpenID. Como se ha comentado más de 9 millones de sitios Web lo soportan pero muchos más son los que no lo han habilitado.

Por ejemplo algunos sitios que tienen habilitado OpenID, se pueden consultar en:

https://www.myopenid.com/directory

La unificación de todas las identidades conlleva altos riesgos, unido a que el servidor de identidad conocerá todas las webs que se visiten cada día. Es más, si se comprometiera la seguridad del ordenador usuario o del servidor, toda la vida digital del usuario quedaría desprotegida.

2.4. Implementaciones de OpenID

2.4.1. PhpMyID

Aplicación web [8] que realiza la funcionalidad de un servidor OpenID, es fácil de instalar y fácil de configurar. Con sólo editar unas pocas líneas en su fichero de configuración y agregar todos los datos del perfil del usuario en el propio código, y ya esta configurado, solo falta acceder a la url para que funcione. El principal problema es que solo se puede utilizar para un único usuario y no tiene mucha seguridad dado que el uso de autentificación es mediante HTTP y lo que hace es encriptar la contraseña con md5 para hacer algo más seguro el transporte de datos.

Pero por temas de seguridad y que además solo se trata de una aplicación para un solo usuario fue descartada esta opción.

2.4.2. Clamshell

Servidor OpenID que permite la gestión de múltiples identidades OpenID desde una sola página web [4] [20]. Se deriva de una serie de fuentes, sobre todo phpMyId, las bibliotecas OpenID JanRain y el uso de Drupal 6.

Los requisitos del servidor son:

- Servidor HTTP como Apache (recomendado)
- PHP 4.3

Lo ventaja de este servidor es que soporta múltiples usuarios pero en su defecto este servidor utiliza HTTP por lo cuál tiene un nivel de seguridad malo y se tendría que poner SSL por debajo. Otra problemática de este servidor es que los perfiles no son modificables, solo se pueden utilizar los campos específicos que implementa la aplicación.

2.4.3. WSO2

WSO2 Identity Server, es un servidor de admistración de código abierto. Es compatible con tarjetas de información y con autenticación OpenID. Proporciona una completa solución de identidad que se integra fácilmente en los servicios de usuarios existentes, tales como LDAP o Active Directory y admite la autenticación de múltiples factores y mucho más. El WSO2 Identity Server [9] [5] ayuda a construir SOA asegurado en unos pocos pasos muy fácilmente.

Como creciente adopción de SOA en una empresa, los recursos SOA, como los procesos y las políticas deben estar bien administrados. WSO2 Identity Server es una solución ideal para responder a los crecientes desafíos de la administración de identidades y seguridad de las aplicaciones Web empresariales. Permite a los arquitectos de la empresa y los desarrolladores mejorar la experiencia del cliente mediante la garantía de las interacciones en línea seguras dentro y fuera de SOA.

El WSO2 Identity Solution permite LAMP y Java en sitios web para proporcionar autenticación fuerte basada en la nueva tecnología de Microsoft CardSpace.

En la versión se incluye OpenID y tarjetas de información, mejorando aún más la solución de identidad WSO2 para atender a un público más amplio para la autenticación basadas en la Web. OpenID es un elemento clave en la descentralización de inicio de sesión único, muy favorecido por muchos usuarios.

La Solución de Identidad WSO2 también trabaja con los actuales directorios empresariales de identidad, tales como los basados en el Lightweight Directory Access Protocol (LDAP) y Microsoft Active Directory, lo que les permite aprovechar su infraestructura existente. Además del proveedor de identidad y de la solución de identidad WSO2, proporciona una parte que confía, y un conjunto de componentes que se conectan a los servicios de la Web más comunes para añadir soporte en la autenticación OpenID y CardSpace.

Funciones clave:

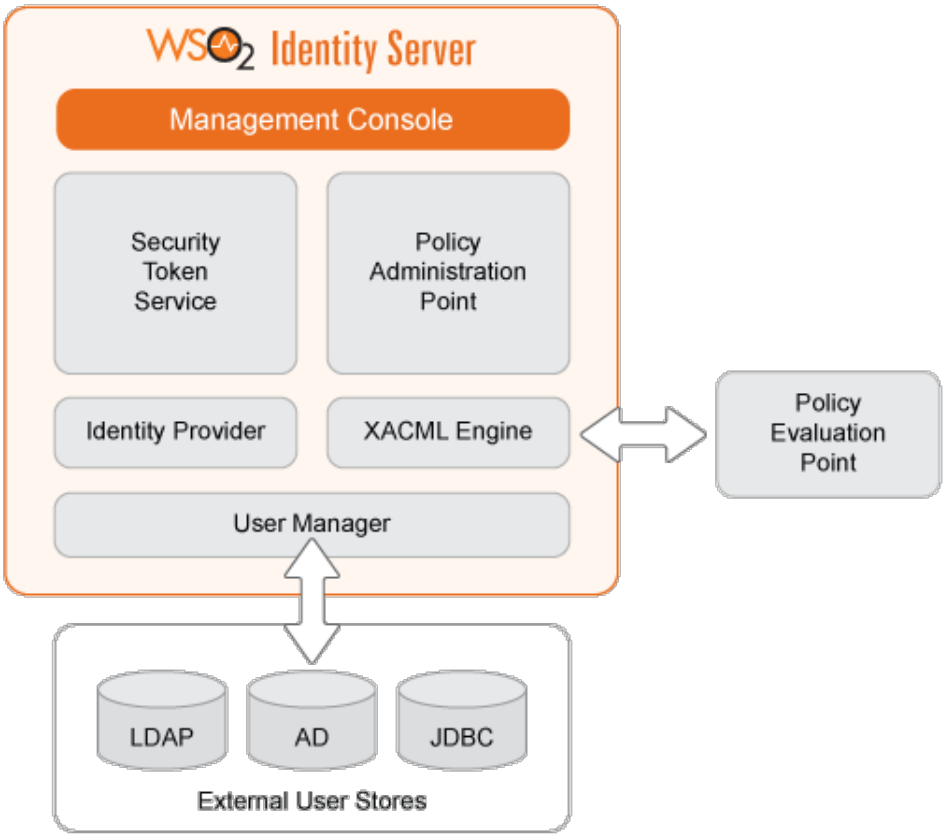

Figura 2.4: Arquitectura de WSO2 Identity Server [9]

- Proveedor de Identidad
  - Gestión simple de la consola.
  - Posibilidad de conectar a las tiendas de usuario personalizada (LDAP / Microsoft ActiveDirectory, JDBC) ActiveDirectory, JDBC).
  - Construido en el almacén del usuario.
  - Apoyo a la demanda que figura por defecto CardSpace.
  - Apoyo a dialectos personalizados y tipos.
  - Estadísticas/reportes.
  - Capacidad para utilizar tarjetas de información.

- Gestiones en las tarjetas de información basada en la credencial de nombre de usuario–token y credencial expedida auto.

- Modulo apache HTTPD con soporte para tarjetas de información relying–party
  - CardSpace admite la autenticación para el contenido web estático.
  - Soporte para cualquier lenguaje de script del lado del servidor con el apoyo de Apache2.
  - Interfaz de fácil integración para desarrolladores.
  - Apoyo a los marcos de gestión de contenidos como Drupal, MediaWiki.
  - Filtro Java Servlet en la parte que confía en componentes.
  - Proporciona una interfaz intuitiva plug–in para desarrolladores de aplicaciones Web J2EE, para habilitar la autenticación CardSpace.
  - Compatible con múltiples valores de respuestas.
  - Compatible con un conjunto de modos de operación sencilla.

2.5. Herramientas

2.5.1. APML

APML (Attention Profiling Markup Language) [3] [16] estándar que promete hacer más fácil para los sitios y los servicios Web el llegar a las preferencias exactas, reduciendo la enorme sobrecarga de información del interminable contenido que hay en la Web.

Provee una forma estandarizada de recoger y valorar la importancia de los datos de atención de tal modo que la información reunida sea útil para los sitios y servicios que se usen en la Web, para que estos puedan ofrecer información y contenido personalizado adecuados a las necesidades particulares.

APML proporciona un lenguaje en el cual se puede tener control personal de la información en un archivo consolidado, para usarlo cuando se crea conveniente o para fines propios. Al mismo tiempo los anunciantes y proveedores de contenido pueden tener un cuadro más claro de lo que podría

interesar, ahorrando la molestia de la publicidad irrelevante que interrumpe y presentando la información y las ofertas a las que vale la pena echar un vistazo.

Cuando se hace uso de un servicio o aplicación que tiene APML habilitado, el perfil de atención se coloca en un archivo APML fácil de compartir, que luego se puede llevar e importarlo en otros servicios que soporten este estándar.

El perfil de atención no solo recoge los datos que interesan en las URL visitadas, las etiquetas que colocas en fotos y vídeos, las canciones que eliges para escuchar, sino que además se puede ver lo interesado que se encuentra en las distintas partes de los contenidos que se visitan.

De modo que si se navega regularmente por ciertas secciones pero otras no interesan, el perfil de atención irá valorando los intereses de acuerdo a esto en el tiempo.

De esta forma los servicios que usan el perfil de atención del usuario para ofrecer contenido relevante podrán hacerlo de acuerdo a temas favoritos, en lugar de otros que no son tan importantes.

Es fácil ver cómo, en el tiempo, esto podría ser una herramienta muy útil para que diferentes servicios tengan una idea más precisa y detallada de los intereses del usuario final, que la que es provista ahora comúnmente por sistemas cerrados empleados por sitios Web específicos.

La idea detrás de APML es comprimir todas las formas de Atención de Datos en un formato de archivo portátil que contiene una descripción de sus intereses.

APML seria el encargado de poder transformar los perfiles a micro formatos para poder importar y exportar los distintos perfiles y así de esta manera fuera más fácil su utilización en sistemas móviles gracias a su portabilidad. Además también era importante que estuviese basado en una especificación para que fuera más fácil su implantación y integración con distintas aplicaciones así como que fuese de ayuda en futuras líneas de investigación.

Esta idea se acabó descartando dado que al parecer se encuentra un poco en desuso, pocos son los sitos y aplicaciones que lo han implantado y

según se puede observar no se está avanzando en su implantación así como en su propio desarrollo. También ayudo a el descarte de esta idea el haber encontrado la herramienta WSO2 Identity server de la que más adelante se habla y contiene una propia estructura en cuanto a formatos para los perfiles.

2.5.2. Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los 90. Con este lenguaje, la empresa tenía como objetivo resolver los problemas aparecidos debido a la proliferación de arquitecturas incompatibles, es decir, diferentes máquinas y diferentes sistemas operativos incompatibles entre si. Esta incompatibilidad dificultaba poder crear aplicaciones distribuidas en una red, como por ejemplo Internet.

La característica de Java [13] [15] es que permite resolver el tema de la incompatibilidad, consiste en que el código fuente de los programas desarrollados en Java, se compila en un lenguaje que no es específico de la plataforma. Este código intermedio es denominado "bytecode" y no puede ser comprendido directamente por la máquina, por lo que tampoco puede ser ejecutado directamente. Para la comprensión y ejecución del código es necesario que sea interpretado por una máquina virtual, la cual en el caso específico de Java recibe el nombre de JVM (Java Virtual Machine), la cual es específica de cada plataforma.

En resumen, las principales ventajas que Java presenta son:

- Lenguaje multiplataforma: El programa desarrollado en Java, una vez compilado, puede ejecutarse en cualquier plataforma que tenga la JVM, ofreciendo de esta manera portabilidad.
- Simplicidad: Java ofrece toda la funcionalidad de un lenguaje potente sin la complejidad presentada por éstos al eliminar la aritmética de punteros, registros (struct) o definición de tipos, entre otras cosas.
- Distribuido: proporciona las librerías y herramientas necesarias para que los programas puedan ser distribuidos, es decir, que puedan interactuar con aplicaciones localizadas en otras máquinas mediante protocolos como por ejemplo Http y ftp.
- Robustez: Java realiza verificaciones en busca de errores o problemas tanto en tiempo de compilación como en tiempo de ejecución. De esta

manera, se pretende detectar los errores lo antes posible en el tiempo de desarrollo. Además, obliga a declarar los métodos y maneja la memoria evitando así preocupaciones sobre la liberación o corrupción de memoria al programador.

- Seguro: La seguridad en Java se ramifica en tres: primero, el verificador de código, el cual verifica el código de bytes a ejecutar para asegurarse de que dicho código se comporta correctamente y no va a ejecutar operaciones no permitidas, segundo, el cargador de clases, responsable de determinar cuándo un applet puede añadir clases a un entorno de Java en ejecución; y tercero, el gestor de seguridad restringe la forma en que un applet puede usar las interfaces visibles de las demás clases y puede realizar comprobaciones de seguridad en tiempo de ejecución sobre métodos potencialmente peligrosos.

A pesar de sus ventajas, también tiene algunos inconvenientes:

- Lentitud: Un lenguaje interpretado siempre presenta como inconveniente una cierta lentitud. Sin embargo, actualmente las JVM utilizan un JIT (Just In Time) Compiler. El JIT se encarga de compilar en tiempo de ejecución el código, minimizando de esta manera el tiempo de ejecución.

- Limitaciones del recolector de basura: éste elemento gestiona la memoria pero en el momento en que actúa no puede controlarse manualmente. No obstante, esta limitación sólo presenta problemática en aplicaciones en tiempo real.

- Limitaciones de la herencia: la herencia múltiple no está permitida. Este problema, afortunadamente, se puede resolver mediante el uso del mecanismo de interfaces.

Dado que java es una gran herramienta, muy amplia y fácil de utilizar, va a ser el lenguaje de programación elegido para la creación de la biblioteca y de la aplicación gráfica.

2.6. Conclusiones

A la vista de las tecnologías presentadas:

Se va a optar por la utilización de java como el lenguaje de programación en la creación de la biblioteca, de la cual utilizaremos las librerías especiales en seguridad como son javax.net.ssl y java.security. Además también se va a utilizar en la aplicación gráfica de prueba, creando un Jframe.

En cuanto a el servidor OpenID, se va a utilizar WSO2 dado que es una completa herramienta que contiene todos los requisitos necesarios como son SSO, SAML, SSL y además permite la integración de almacenes de usuarios externos, introducción de políticas, inclusión de certificados, modificación de los perfiles de forma sencilla, posibilidad de introducir nuevos campos en los perfiles así como la posibilidad de disponer de esquemas de los distintos atributos de los perfiles.

WSO2 es la mejor herramienta en comparación con las otras opciones expuestas (PhpMyID y Clamshell) y que ofrece mayores opciones y características, de la cual se pueden sacar más características, las cuales pueden servir en trabajos futuros.

Capítulo 3

Diseño de la biblioteca

3.1. Introducción

A lo largo de este apartado se citarán y explicarán cada una de las decisiones de diseño que se han tomado durante la realización del proyecto para conseguir la comunicación deseada entre la biblioteca y el servidor.

La biblioteca va a ser la encargada de recuperar y enviar datos entre el usuario y el servidor así como interactuar con estos mismos datos en una aplicación o servicio, para poder trabajar sin necesidad de que estos accedan al propio servidor y así facilitar la integración de contenidos y la posibilidad de poder realizar esta tarea automáticamente con varias aplicaciones o servicios a la vez.

Para ello esta biblioteca tiene que ser capaz de poder interactuar con el servidor y poder tanto recuperar como agregar contenidos, para el posterior tratamiento de estos datos y para poder trabajar con estos datos sin conexión.

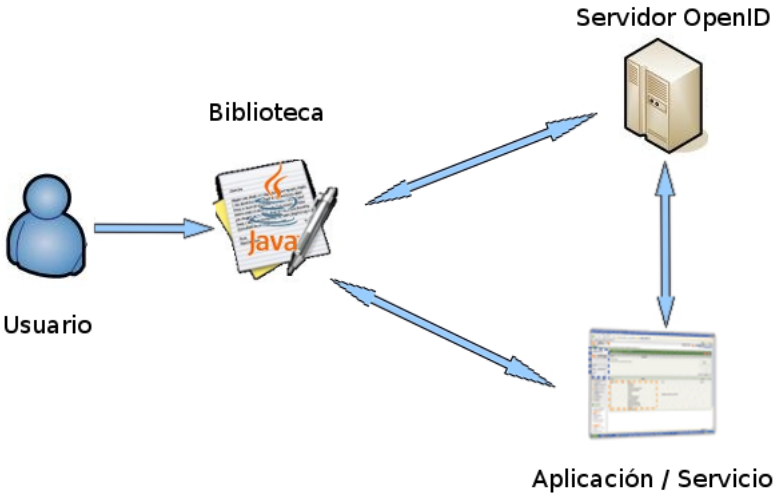

Figura 3.1: Diagrama de acceso

3.2. Descripción de las funcionalidades de la biblioteca

Esta biblioteca lo que pretende es facilitar la portabilidad de los perfiles alojados en los proveedores de identidad, para ello se conectara de forma transparente para el usuario, y dará la posibilidad de interactuar con los contenidos alojados en el servidor del proveedor de identidad.

La biblioteca está organizada en una sola clase que es el conector, en está clase se encuentran descritos los distintos métodos que facilitan la integración entre distintas aplicaciones o usuarios y la parte servidora.

Contiene métodos elementales de conexión como es Conectar() que realiza la propia conexión con el servidor mediante los parámetros indicados en el constructor y encargados de comprobar que se puede acceder al servidor y obtener las cookies.

Luego se han integrado métodos de autentificación de usuarios, creación de nuevos usuarios y de cierre de sesión de los usuarios. Este proceso se realiza de forma segura mediante SSLv3 para asegurar que no se produzcan ataques de seguridad entre las aplicaciones y que se envíen los datos de forma segura, estos métodos también gestionan la función de mantenimiento

de conexión mediante el uso de las cookies que se obtienen del servidor.

Las funcionalidades básicas de la aplicación son las que interactúan mediante los perfiles, buscando posiciones, valores, leyendo perfiles y modificando sus valores, estas son totalmente intuitivas de utilizar.

También se ha integrado la posibilidad de poder agregar certificados y consultar los certificados existentes en el servidor así como la posibilidad de eliminarlos.

Los eventos de errores que se produzcan, están definidos para que puedan ser capturados por las aplicaciones y así poder tener más información acerca de fallos y hacer más fácil el desarrollo de aplicaciones futuras.

3.3. Formato de los perfiles

Para el formato de los perfiles se decidió utilizar APML pero tal y como se comento anteriormente, esta idea se desecho y se decidió utilizar otro formato, en su caso se decidió seguir con la estructura que utiliza el servidor WSO2, el cual es un fichero XML con la estructura de los perfiles, en los cuales se indica la sintaxis de los valores permitidos, así como descripciones de los atributos y posición de importancia. Este formato facilita en gran medida la integración con otras aplicaciones dado que así hay un lenguaje y una estructura en común, la cual puede ser consultada en cualquier momento.

El esquema XML:

El documento esta compuesto por múltiples dialects, que estos van a ser cada uno de los campos que contiene el perfil, en el dialect se especifica una URL, y en esta URL será de donde se obtenga la propia estructura de los datos.

Luego están las claims que son los distintos atributos de los perfiles, y dentro de estos se encuentra la claim del propio atributo si la tiene, el nombre del atributo, el identificador, una descripción del atributo, si es requerido o no, la posición en la cual se va a mostrar y si va a estar soportada por defecto.

Ejemplo de un esquema:

<Dialects\>

<Dialect dialectURI=”http://wso2.org/claims/givenname”>

<Claim>

<ClaimURI>http://wso2.org/claims/givenname</ClaimURI>

<DisplayName>First Name</DisplayName>

<AttributeID>http://wso2.org/claims/givenname</AttributeID>

<Description>First Name</Description>

<Required/>

<DisplayOrder>0</DisplayOrder>

<SupportedByDefault/>

</Claim>

</Dialect>

</Dialects>

3.4. Interconexión

El principal objetivo de la biblioteca es ofrecer la portabilidad de perfiles gracias a la posibilidad de poder tener todos los servicios que ofrece los proveedores de identidad al usuario como son el poder registrarse, ver, modificar y eliminar perfiles así como agregar certificados de confianza.

Para poder realizar esas acciones, la biblioteca tiene que interactuar con el servidor de identidad, en el siguiente diagrama se muestra como es una típica petición de datos:

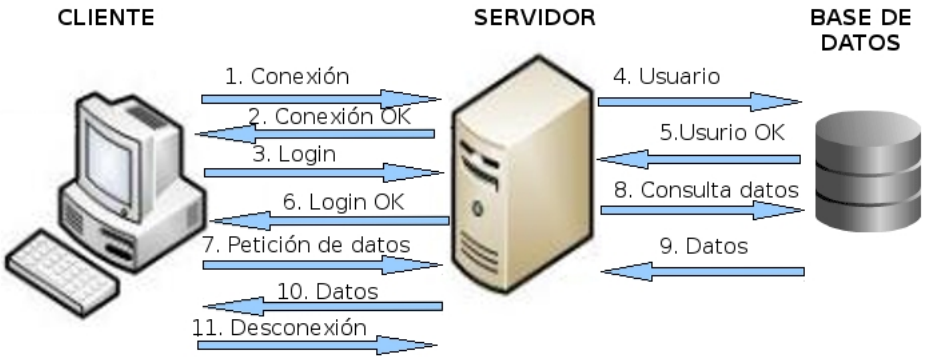

Figura 3.2: Diagrama de conexión

Primeramente se realiza una conexión con el servidor, una vez conectado, se realiza la autentificación del usuario, contra el servidor y este a su vez tiene que consultar su existencia en la base de datos, una vez resuelta, ya se pueden hacer peticiones de los datos del usuario, que el servidor realizara sobre la propia base de datos y por último simplemente queda cerrar la conexión.

La biblioteca se va a conectar al servidor estableciendo una conexión https, esta conexión se realiza de forma segura mediante SSLv3 para asegurar que no se produzcan ataques de seguridad e así iniciar una sesión con el servidor mediante javax.net.ssl [2] y java.security [1], para la obtención del certificado de seguridad del propio servidor. Una vez se ha accedido al servidor y se ha obtenido el certificado, ya se puede hacer el registro de los usuarios, mediante el intercambio de cookies.

Esta biblioteca, principalmente lo que realiza es la descarga del contenido html de la web [10], después almacena las cookies y va leyendo la información que recibe. Mediante esa información envía respuestas utilizando POST que estos serán los datos que se quieran enviar al servidor, también se envían y reciben las cookies si son modificadas para poder mantener la conexión con el usuario que se ha registrado.

3.5. Diseño de la biblioteca

Se opto por la creación de una única clase, dado que al principio se pensó en crear otras clases que trabajaran sobre los perfiles estando guardados en algún fichero y se vio que no era necesario realizarlo dado que era más eficiente trabajar directamente contra el servidor, dado que las peticiones de datos no se demoran mucho y es más eficiente que tener que trabajar con ficheros en disco, para ir leyendo y escribiendo en todo momento, por ello se ha decidido crear una única clase que se encargue de ir enviando y recibiendo datos en todo momento.

Esta clase esta compuesta por unos pocos atributos necesarios para poder establecer y mantener la conexión como son el nombre del servidor, El identificador OpenID, la contraseña, la url que se utiliza en cada momento y las cookies para su almacenamiento mientras se esta conectado.

El conjunto de estos métodos y su interconexión, lo que logran es poder facilitar el manejo y recuperación de datos para que un usuario que implemente esta biblioteca sea capad de poder utilizar todas las características que ofrece el servidor de una manera muy cómoda y sencilla.

Acontinuación se muestra la estructura que tiene la clase y donde se pueden apreciar los atributos y metodos implemetados para lograr el objetivo deseado.

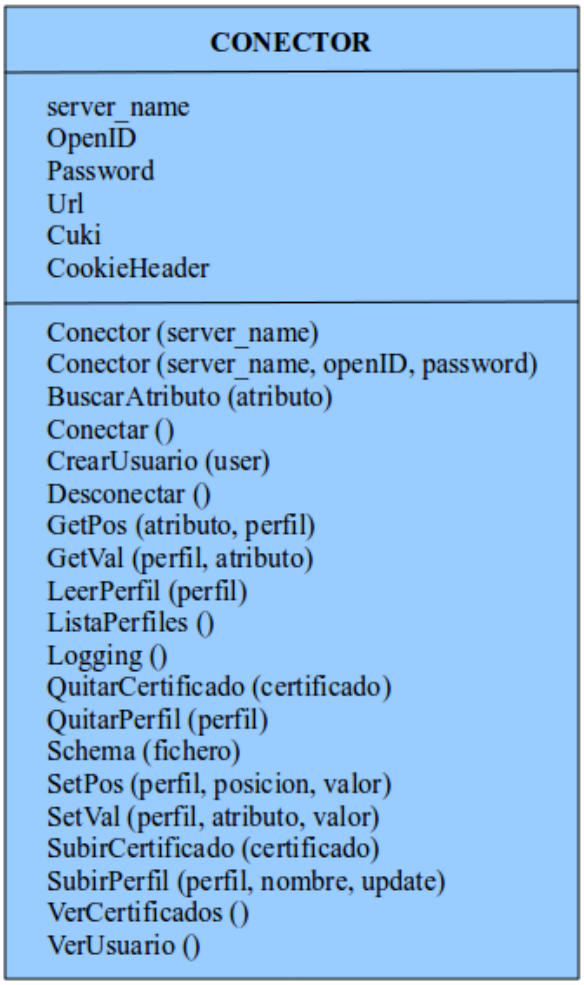

Figura 3.3: Diseño UML de la clase Conector

3.5.1. Obtener datos del servidor

Para la Obtención de los datos del servidor se van a crear distintos métodos para facilitar la recuperación de los datos. Estos métodos van a ser capaces de navegar por el servidor y recuperar el código fuente de los datos que se necesiten.

Después de la recuperación del cogido fuente, se filtraran los datos para poder sacar información relevante para su posterior tratamiento.

Los métodos de recuperación de datos implementados, que se encargan de sacar la información de los perfiles y los certificados son los sigu-

ientes: (LeerPerfil(perfil), ListaPerfiles(), VerCertificados(), VerUsuarios(), Schema(fichero)).

3.5.2. Manipular perfiles

En la manipulación de los perfiles, se va a proceder a acceder un POST de los datos que se quiera modificar, este post se va a realizar con los contenidos del formulario que el servidor desea recibir y los cuales se realizan en determinadas URLs dependiendo de los datos a modificar.

Métodos de envío de datos: son los que modifican o agregan información en el servidor (CrearUsuario(user), SetPos(perfil, posición, valor), SetVal(perfil, atributo, valor), SubirCertificados(certificado), SubirPerfil(perfil, nombre, update)).

Métodos de eliminación de datos: eliminan los perfiles o certificados que se deseen (QuitarCertificado(certificado), QuitarPerfil(fichero)).

Además para facilitar la utilización de los métodos internos y de los futuros desarrollos se han creado métodos de búsqueda de datos que recuperan datos del servidor y buscan entre ellos los datos deseados.

Métodos de búsqueda: buscan dentro de perfiles atributos, valores y posiciones de datos (BuscarAtributo(atributo), GetPos(atributo, perfil), GetVal(perfil, atributo)).

3.6. Aplicación grafica

La Aplicación gráfica esta basada en un Jframe para la comprobación total de la funcionalidad de la biblioteca, la cual implementa la clase conector.java. Está aplicación se ha basado en la parte servidora de donde obtiene todas las características que residen en ella.

La aplicación simula todas las posibilidades que se pueden contemplar como son la creación de un usuario, así como poder autentificarse con un usuario y poder cambiar todas las opciones posibles, en los perfiles.

La aplicación va a estar compuesta por varios modulos:

En la pantalla principal aparecerá la opción de autenticación o la de registrarse.

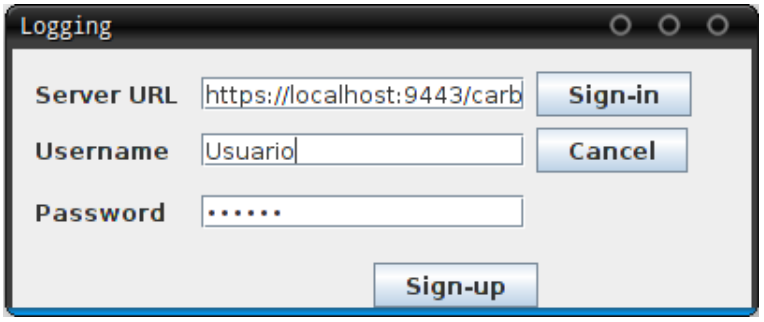

Figura 3.4: Autenticación

El registro de usuarios, será una pantalla en la cual se pedirá introducir los datos del usuario que se consideren importantes para el registro, prestando mayor atención en los campos con * que son obligatorios. Este perfil se realizara sobre una plantilla por defecto, la cual luego se podrá modificar.

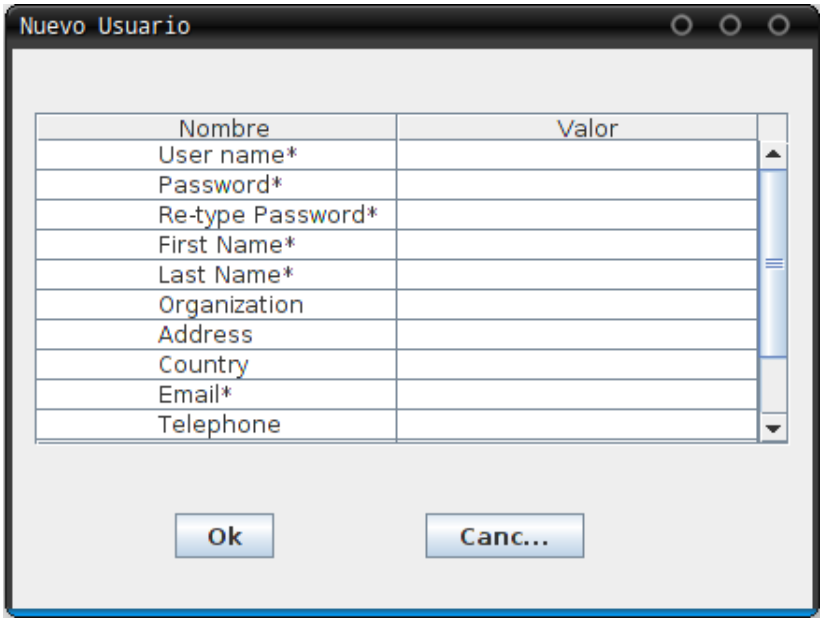

Figura 3.5: Creación de un nuevo usuario

Una vez creado el usuario, éste ya se puede autenticar en la aplicación y en ella se deberá introducir el proveedor de identidad, el usuario y la contraseña, para poder acceder al servicio.

Cuando el usuario accede al portal con sus datos, accedera a otra pantalla que sera su propio perfil.

El perfil, será otra pantalla JFrame en la que el usuario, podrá visualizar todos sus perfiles existentes mediante un selector de perfiles y en el que además se dará la opción de poder agregar nuevos perfiles, así como de modificarlos e eliminarlos. También estará disponible una opción de generar el esquema xml para poder almacenar los datos en local o para uso de otras aplicaciones, se indica en donde se quiere que se guarde el documento XML, introduciendo la ruta y el nombre del fichero con extensión xml.

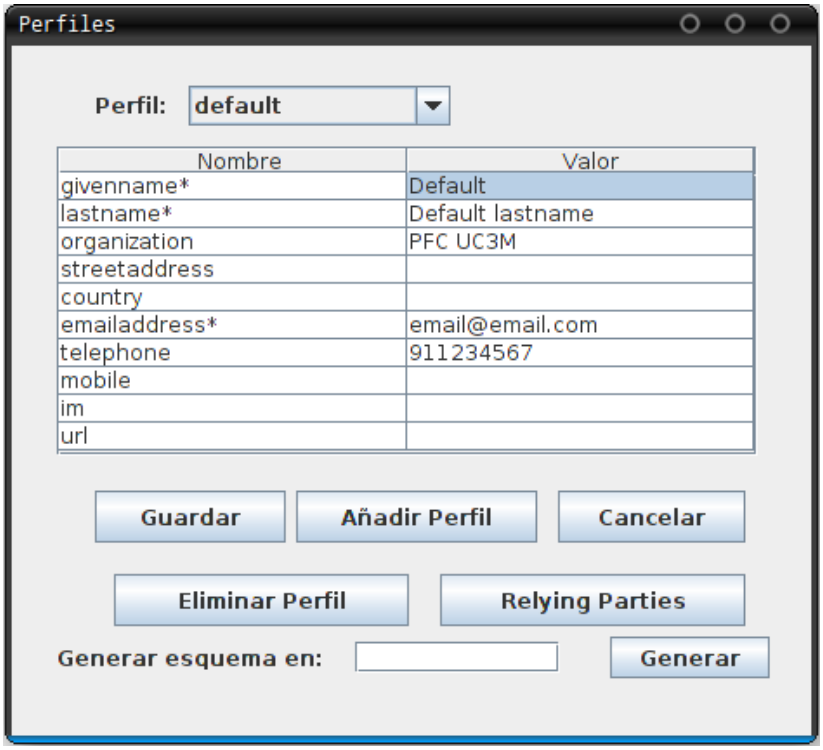

Figura 3.6: Pantalla principal donde se muestra el perfil

Relying parties, pantalla en la cual el usuario va a poder ver los certificados existentes, así como agregar nuevos mediante un explorador de ficheros o eliminar certificados ya existentes en la base de datos con el simple echo de seleccionarlos y pulsar sobre el botón eliminar.

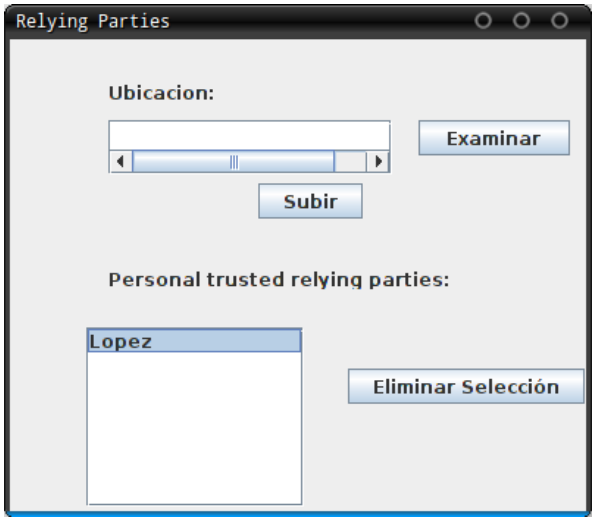

Figura 3.7: Visualización de certificados existentes

Capítulo 4

Implementación

4.1. Biblioteca

Una vez definidas las soluciones de diseño llega el momento de implementar los métodos que permitirán llevar a cabo todas estas funciones.

Los métodos de conexión, se realizan estableciendo una conexión SSL a la url y se guardan las cookies para ir manteniendo la sesión que se ha establecido.

En el caso de envió de datos al servidor, lo que se hace es descargarse el código fuente de la url donde se encuentra el POST que se quiere enviar, y una vez obtenido lo que se hace es completar los campos requeridos y enviarlos.

Como por ejemplo podemos comprobar que al obtener el código fuente de la pagina de inicio del servidor podemos encontrar entre el código los datos necesario para enviar el POST de autentificación al servidor.

```
<form action='login_action.jsp' method="POST" target="_self">
    <table>
        <tr>
            <td>
                <nobr><label for="txtUserName">Server URL</label></nobr>
            </td>
            <td>
                <input type="text" id="txtbackendURL" name="backendURL"
                       class="user" tabindex="1" value="https://localhost:9443/services/"/>

            </td>
        </tr>
        <tr>
            <td>
                <label for="txtUserName">Username</label>
            </td>
            <td>
                <input type="text" id="txtUserName" name="username"
                       class="user" tabindex="1"/>

            </td>
        </tr>
        <tr>
            <td>
                <label for="txtPassword">Password</label>
            </td>
            <td>
                <input type="password" id="txtPassword" name="password"
                       class="password" tabindex="2"/>

            </td>
        </tr>
        <tr>
            <td> </td>
            <td>
                <input type="submit" value="Sign-in"
                       class="button" tabindex="3"/>
            </td>
        </tr>
    </table>

</form>
```

Por lo tanto solo se tiene que hacer un POST a la URL de conexión:

```
url = new URL(server_name + "admin/login_action.jsp");
HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
conn.setRequestProperty("Cookie", cookieHeader);
conn.setRequestMethod("POST");
```

Se rellena el POST con los datos necesarios codificados en UTF–8:

```
OutputStreamWriter osw = new OutputStreamWriter(conn.getOutputStream());
osw.write(URLEncoder.encode("backendURL", "UTF-8") + "=" + URLEncoder.encode(server_name.replaceAll("carbon/",
    "") + "services/", "UTF-8"));
osw.write("&");
osw.write(URLEncoder.encode("username", "UTF-8") + "=" + URLEncoder.encode(openID, "UTF-8"));
osw.write("&");
osw.write(URLEncoder.encode("password", "UTF-8") + "=" + URLEncoder.encode(password, "UTF-8"));
osw.write("&");
osw.write(URLEncoder.encode("submit", "UTF-8") + "=" + URLEncoder.encode("Sign-in", "UTF-8"));
osw.flush();
osw.close();
conn.setInstanceFollowRedirects(false);
```

Y ya solo queda almacenar las cookies para continuar con el misma autentificación.

cookieHeader = conn.getHeaderField(”set–cookie”);

Los métodos que recuperan los datos de los perfiles o las estructuras de los perfiles se encargan de descargar el código fuente de dichas secciones y sacan los datos requeridos dado que se conoce la estructura html que emplea el servidor para ello y lo que hacen es buscar dentro del html los datos de la estructura e ir almacenando en arrays de strings los datos que hacen falta.

Por ejemplo para recuperar los perfiles existentes, lo primero es entrar en la url que los contiene:

```
URL url2 = new URL(server_name + "userprofile/index.jsp?region=region5&item=userprofiles_menu&ordinal=0");
HttpsURLConnection conn2 = (HttpsURLConnection) url2.openConnection();
```

Se descarga el código fuente:

```
InputStream ins = conn2.getInputStream();
InputStreamReader isr = new InputStreamReader(ins);
BufferedReader in = new BufferedReader(isr);
```

Una vez descargado, se busca entre la estructura del código fuente los datos necesarios.

```
<div style="height:30px;">
        <a href="javascript:document.location.href='add.jsp?username=Alberto&print=false'" class="icon-link"
           style="background-image:url(../admin/images/add.gif);">Add New Profile</a>
    </div>
       <table style="width: 100%" class="styledLeft">
               <thead>
                       <tr>
                               <th colspan="2">Available Profiles</th>
                       </tr>
               </thead>
                       <tbody>
               <tr>
                       <td width="50%"><a href="edit.jsp?username=Alberto&profile=pepe&userstore
                            =Internal&print=false">pepe</a></td>
                       <td width="50%"><a title="Remove Profile"
                          onclick="remove('Alberto','pepe');return false;"
                          href="#" style="background-image: url(../userprofile/images/delete.gif);"
                                class="icon-link">
                           Delete</a></td>
               </tr>
               <tr>
                       <td width="50%"><a href="edit.jsp?username=Alberto&profile=default&userstore
                            =Internal&print=false">default</a></td>
                       <td width="50%"><a title="Remove Profile"
                          onclick="remove('Alberto','default');return false;"
                          href="#" style="background-image: url(../userprofile/images/delete.gif);"
                               class="icon-link">
                           Delete</a></td>
               </tr>
         </tbody>
         </table>
  </div>
```

Figura 4.1: Código fuente de los perfiles existentes

Para buscar se miran las etiquetas html para poder identificarlo y una vez encontrado se almacena.

```
while ((line = in.readLine()) != null) {
            if (line.contains("Available Profiles") || i == 1) {
                if (line.contains("<a href=")) {
                    String ad = line.replaceAll("\"", "");
                    ad = ad.substring(ad.indexOf("&print=false>"));
                    ad = ad.replaceAll("&print=false>", "");
                    ad = ad.replaceAll("</a></td>", "");
                    perfil[c] = ad;
                    ++c;
                }
                i = 1;
            }
            if (line.contains("/table")) {
                i = 0;
            }
        }
    }
```

Los métodos que envían datos, primeramente se dedican a acceder a la sección de modificación o creación de perfiles y una vez allí, si es necesario

como en el caso de creación de usuarios, se descarga el formulario y a continuación se rellena el formulario web de creación de usuario y se envía al servidor mediante un POST.

Los método de búsqueda, están implementados para que llamen al método de LeerPerfil(perfil) de uno o varios perfiles y una vez se tienen los perfiles, se van recorriendo los datos de los mismos hasta encontrar una coincidencia con la búsqueda y en este caso se devuelve la posición en la que se encuentran estos datos o el perfil que los contiene.

Para poder subir certificados se realiza mediante POST en los cuales se añaden cabeceras especiales que requiere el servidor y entre las cabeceras se introduce el certificado en texto plano.

```
String lineEnd = "\r\n";
String twoHyphens = "--";
String boundary = "***232404jkg4220957934FW**";
int bytesRead, bytesAvailable, bufferSize;
FileInputStream fileInputStream = new FileInputStream(new File(certificado));
conn.setRequestProperty("Content-Type", "multipart/form-data;boundary=" + boundary);
dos = new DataOutputStream(conn.getOutputStream());
dos.writeBytes(twoHyphens + boundary + lineEnd);
dos.writeBytes("Content-Disposition: form-data; name=\"rpcert\"; filename=\"" + certificado + "\"" + lineEnd);
dos.writeBytes("Content-Type: application/octet-stream" + lineEnd);
dos.writeBytes(lineEnd);
```

Una vez se rellena el formulario se procede al envío de los datos, en este caso como son muchos datos se utiliza el buffer para poder realizar el envío.

El código de la biblioteca se encuentra dentro de la carpeta src, dentro de este directorio podemos ver otra carpeta llamada aplicacion_pfc y en el interior de ese directorio se allá la clase Conector.java que es la propia biblioteca en si y donde se puede consultar su código, así como la posibilidad de editarlo para trabajos futuros.

En el directorio dist podemos ver la carpeta javadoc que es donde está contenida la documentación de toda la biblioteca. En ella podemos consultar todas y cada una de las funcionalidades que podemos utilizar de la biblioteca. En cada método se indica claramente todas y cada una de las funcionalidades que contiene así como los distintos parámetros que requieren los métodos y su estructura.

4.2. Aplicación gráfica

La aplicación gráfica esta basada en JFrame, consta de cuatro clases:

1. Main.java.

   Encargado de ejecutar la aplicación.

2. Aplicacion.java

   Es la pantalla principal donde se pide al usuario que se autentique o que se registre.

3. NewUser.java.

   JFrame que contiene un formulario con el cual se ofrece la posibilidad de crear un nuevo usuario OpenID con un perfil básico.

4. Perfil_pan.java.

   Es el panel principal donde se muestran los diferentes perfiles de los usuarios.

La primera y más importante el Main que es la encargada de ejecutar la clase Aplicación al iniciar y hacerla visible.

La Aplicación, es el cuestionario de looging y en el que se muestran dos JTextField para completar los campos de la URL del servidor y el nombre del usuario. Para la contraseña se utiliza un JPasswordField que se encarga de no mostrar la contraseña cuando el usuario la escribe. También se dispone de tres botones, uno para loguearse contra el servidor, que lo que hace es llamar a la al método Logging() de la biblioteca. El segundo botón, es para cancelar y salir, y el tercer botón es para la creación de un nuevo usuario.

La clase NewUser es la encargada de mostrar un JTable donde se introducen los datos necesarios para crear un nuevo perfil, se dispone de dos botones, uno para aceptar una vez terminado y el cual llama al método CrearUsuario() de la biblioteca, si todos los datos requeridos están completados y en el formato adecuado, de lo contrario en la parte de abajo se mostrara un mensaje de error. El otro botón realiza la operación de cancelar y salir de la aplicación.

La clase Perfil_pan es la más completa y es la que muestra los perfiles del usuario que se podrán mediante un JCombobox y según se vayan seleccionando, en la JTable de abajo se irán mostrando. Los botones que ofrece

son para interactuar con el perfil, llamando a los métodos de modificación, creación y eliminación de perfiles que tienen la biblioteca, y los cuales interactúan con la JTable. Además se dispone de un JTexfield, en el cual se tienen que poner el destino y el nombre del fichero para si se quiere descargar un esquema de los perfiles que se disponen. El otro botón de Relying Parties, muestra otro JFrame, en el cual se pueden subir certificados, gracias a un JFileChooser y además se pueden visualizar los certificados existentes en el servidor en una JList, de donde se pueden seleccionar los elementos he eliminar, mediante el botón de eliminar Selección.

Para ejecutar la aplicación solamente se a de introducir el siguiente comando en una línea de comandos en un sistema en el cual este instalado java.

java –jar Aplicacion_PFC/dist/Aplicacion_PFC

Toda la documentación de está aplicación se encuentra dentro de la carpeta javadoc dentro del directorio dist, en ella se puede consultar todos los métodos implementados.

4.3. Entorno de desarrollo WSO2 Identity server

Para la implementación del proveedor de identidad se va a hacer uso de la herramienta WSO2 Identity server, mediante la cual se puede comprobar el correcto funcionamiento de la biblioteca.

Lo primero es realizar la instalación de dicho servidor en una maquina que se encuentre en la red donde se va a utilizar la biblioteca como puede ser en internet o en una red privada.

A continuación se crea un dominio o si ya se dispone de uno no hace falta crearlo, también esta la posibilidad de poder utilizar una ip estática en su lugar, pero es más difícil de memorizar. Este dominio será el que de nombre al servidor OpenID y el cual va a ofrecer el servicio.

Para poder instalar el servidor, primeramente se tiene que instalar java, dado que el servidor esta implementado en esta tecnología.

Después se puede realizar la instalación del servidor, la cual se puede realizar mediante los pasos que se indican en el Apéndice A. A la hora de la instalación se tiene que introducir un número de puerto, el cual si se quiere que salga fuera de nuestra red se tiene que habilitar en el corta fuegos del router o equipo de interconexión, este paso es importantísimo dado que sin el no se podría accede al servidor desde fuera de la red.

Una vez instalado se puede hacer uso del Apéndice B donde se indican los pasos a seguir para que el servidor reconozca el host y el puerto de dicha maquina y así poder trabajar con el nombre del dominio que se tenga.

Otra cosa importante a realizar es la creación de un certificado SSL propio ya que el servidor trae uno predeterminado y no es adecuado para un entorno de producción y seria conveniente crear un nuevo certificado con los datos del host y la organización, esta acción esta explicada en el apéndice C donde se detalla con un claro ejemplo como se puede crear este certificado e introducirlo en el servidor.

Cuando ya este el servidor activo, ya se podrán crear usuarios, así como configurar las múltiples opciones que ofrece el servidor.

En el apéndice D se incluye un pequeño manual para usuarios en el que se indica cuales son las configuraciones que se pueden hacer sobre la cuenta y las herramientas que ofrece como son la posibilidad de la utilización de tarjetas de información con los propios datos del usuario así como mejoras en la autenticación gracias a XMPP.

En el apéndice E se detalla como se puede administrar el servidor con todas las posibilidades que este brinda como son la posibilidad de introducir políticas, gestionar usuarios, gestionar perfiles, introducir certificados externos, configurar la emisión de tarjetas de información y poder ver las estadísticas y sucesos que se van produciendo al instante.

En el apéndice F también se detalla como poder añadir almacenes de usuarios externos, para poder reutilizar algún almacén de usuarios ya existentes y no tener que dar un nuevo servicio a todos los usuarios.

Una vez configurado el servidor al gusto, ya es posible la interacción con el y así poder utilizar la biblioteca.

Capítulo 5

Pruebas

5.1. Introducción

En primer lugar se han realizado pruebas de caja blanca, que son pruebas software que se realizan sobre las funciones internas de un módulo, de manera que se consigan cubrir todo el código del programa comprobando los posibles errores.

En segundo lugar se han realizado pruebas de caja negra, que son pruebas funcionales que se realizan sobre el exterior de un módulo, en las que se aporta una entrada y se comprueba la salida, sin importar lo que haya ocurrido dentro del módulo.

A continuación se muestra el plan de pruebas seguido:

5.2. Pruebas de caja blanca

Durante la etapa de desarrollo, el programa se ha compilado diversas veces, comprobando los resultados obtenidos tras su ejecución.

Las pruebas iniciales se realizaban con pequeñas conexiones al servidor para comprobar que realmente se estaba tanto accediendo a los contenidos disponibles en el servidor, como pudiendo enviar datos y que las respuestas obtenidas en ambos casos fuesen coherentes.

Gracias a que el servidor WSO2 dispone de la posibilidad de poder ver los logs que se generan en cada instante, podemos ir viendo cuando se van produciendo eventos de conexión y desconexión, y así poder determinar que

todas las peticiones que estamos realizando, se van cumpliendo satisfactoriamente.

Un sistema utilizado es la inclusión de trazas en el código para poder ir indicando que va ocurriendo en cada momento y ver cuáles son los errores que se van generando para poder detectar fallos en el sistema y con ello facilitar el poder localizar errores.

Para la Inserción de datos así como puede ser la modificación y eliminación de datos, se utilizaban dos métodos de comprobación, el primero se encargaba de descargar los datos que se encontraban en el servidor y el otro consiste en introducirse directamente en el propio servidor y verificar que todo se ha realizado correctamente.

Las pruebas de la biblioteca se fueron haciendo mediante pequeñas ejecuciones que se realizaban en cada método para ir viendo que todo estaba funcionando correctamente y que se obtenía una respuesta adecuada en todo momento, tanto cuando se realizaba una acción correcta como incorrecta, si en algún caso no se obtenía respuesta se llegó incluso a analizar el tráfico de datos que se intercambia entre la biblioteca y el servidor para comprobar que todo se realizaba con total normalidad.

Una vez terminada la aplicación, esta ha sido sometida a nuevas pruebas, entre ellas se han realizado pruebas introduciendo siempre datos ficticios e incorrectos para ver que eran detectados como fallos.

5.3. Pruebas de caja negra

A continuación se empezó a probar la biblioteca mediante la realización de una aplicación gráfica que simulaba el total comportamiento que puede tener un usuario en el servidor.

En la aplicación grafica se ha podido comprobar como se puede realizar la autentificación y en caso de que este no se pueda realizar correctamente, dado que alguno de los campos sea erroneo, se devuelva un mensaje al usuario avisando acerca de lo que esta sucediendo, tanto si se introduce una url invalida, como si el usuario y la constaseña no son validos. También se ha probado que la contraseña no se muestra en ningún momento de la transacción.

En el registro de usuarios se ha podido comprobar como la funcionalidad de los campos requeridos funcionan y no deja crear un usuario sin tener todos los datos completados, Además se ha podido observar como cuando se rellenan los campos estos llegan idénticamente al servidor, con todos los campos iguales y con todos los caracteres especiales incluidos. También se ha comprobado la estructura de los campos como puede ser el correo que tiene una estructura especial o el teléfono que esta compuesto solo por números.

Otro caso importante del registro es la imposibilidad de poder crear usuarios con el mismo nombre.

El perfil, se puede observar que obtiene todos los perfiles existentes con todos los campos y que estos se pueden modificar y son directamente modificados en el propio servidor.

Se probo la funcionalidad de poder crear nuevos perfiles de todas las formas, ya sea vacios o con campos erróneos como por ejemplo el correo y también se probo la introducción de caracteres especiales.

En el caso de los certificados se comprobó que se podían recuperar los nombres de los certificados existentes y la posibilidad de subir los certificados satisfactoriamente y que solo aceptase la extensión .cert. También se pudo comprobar que a la hora de borrar un certificado, también realizaba la operación en el propio servidor.

Capítulo 6

Presupuesto

El presupuesto de este proyecto consta de dos partes, la primera es el coste del personal, para el cual se contratara a un desarrollador durante 3 meses realizando 8 horas al día. Por otra parte esta el coste del material utilizado en el desarrollo del proyecto.

- Coste Personal:

  Días laborables / mes: 20.
  Horas trabajadas / día: 8 horas.
  Horas trabajadas / mes: 160 horas.
  Salario neto por persona al mes: 1200 euros.
  IRPF (20 % sobre el salario neto): 240 euros.
  Seguridad Social (40 % sobre salario neto + IRPF): 576 euros.
  Salario bruto por persona / mes: Salario neto + IRPF + Seguridad Social: 2016 euros.
  Meses laborables: 3 meses.
  Salario bruto por persona (3 pagas): 6048 euros.

- Coste Equipamiento:

  Equipo informático (PC) 700 euros.
  Software libre sin necesidad de licencias, tanto java como WSO2.
  Terminal móvil para pruebas 250 euros.
  Linea ADSL 45 x 3 meses = 135 euros.

- Coste Total:
  - Costes indirectos: obtenidos como el 15 % de los costes directos, esto es el 15 % sobre coste de personal + equipamiento.
  - Riesgo: en el proyecto actual, es el personal el que presenta mayor riesgo, por lo que se calcula como el 20 % sobre el coste de personal.
  - Beneficio: el 25 % del coste total, entendido este como la suma de los costes directos + costes indirectos + riesgo.

Descripción	**Coste (Euros)**
Coste de personal	6048
Equipos informáticos y licencias	1085
Costes directos = Coste de personal + Equipos	7133
Costes indirectos = 15 % sobre Costes directos	1069.95
Riesgo = 20 % sobre Coste de personal	1209.6
Total costes = Costes directos + Costes indirectos +Riesgo	9412.55
Beneficio = 15 % sobre Total costes	1411.88
TOTAL = Total costes + Beneficio	**10824.43**

Cuadro 6.1: Presupuesto del proyecto

Capítulo 7

Historia del proyecto

En este capítulo se describen las diferentes fases en las que se ha desarrollado este proyecto de fin de carrera, así como la duración de cada una de las fases.

7.1. Fases del proyecto

- FASE I: Documentación.

  Los objetivos del proyecto comprendían el acceso y manipulación de los perfiles de usuario (identidades digitales) repartidas en diferentes proveedores de identidad. Debido a esto, encontrar un formato y un lenguaje con el que almacenar y manipular dichos perfiles era clave para el proyecto.

  En principio se centro la atención en APML, mencionado en el estado de la técnica. El concepto perseguido por APML era el deseado, pero dicho lenguaje se encuentra en desuso y esto motivo la evaluación de un formato similar al que emplea openID para caracterizar los atributos.

- FASE II: Búsqueda de proveedores de identidad.

  Se estuvo bastante tiempo buscando proveedores de identidad y evaluando sus características, Se llego a la conclusión que el más adecuado para el proyecto era el uso de OpenID.

Después se empezó a buscar y a probar distintos servidores OpenID que se adaptaran a nuestras necesidades como fueron CLAMSHELL, PHPMYID, WSO2..., todos ellos se instalaron y probaron y de los cuales, al final se pudo comprobar que el que más se acerca a nuestras necesidades era WSO2 por los motivos anteriormente indicados.

- FASE III: Instalación de la parte servidora.

  En esta fase se realizo la instalación del servidor OpenID WSO2 y se continúo investigando sobre sus características y la configuración de las distintas características que este ofrece.

- FASE IV: Entorno de desarrollo.

  Se pensó ampliamente en cual era el mejor entorno para la realización de la biblioteca y se llego a la conclusión de que Java ofrecía una mayor flexibilidad como mayor portabilidad de dispositivos, así como una extensa API de la cual se podía hacer uso.

- FASE V: Realización de la Biblioteca.

  A continuación se empezó a diseñar la biblioteca, se empezó buscando una estructura y unas funcionalidades que luego se empezaron a implementar.

- FASE VI: Pruebas y creación de aplicación de prueba.

  Luego se realizaron ciertas pruebas sobre la biblioteca y finalmente se creó una aplicación gráfica donde se podía ver el completo funcionamiento de la biblioteca, así como la detección fallos y de posibles mejoras.

- FASE VII: Memoria.

  Esta fase consiste en la documentación de todo el trabajo realizado, acción materializada en la presente memoria.

Fase	Descripción	Duración
Fase I	Documentación	1 mes
Fase II	Búsqueda de servidores OpenID	2 semanas
Fase III	Instalación de la parte servidora	1 mes
Fase IV	Entorno de desarrollo	1 semana
Fase V	Realización de la Biblioteca	2 meses
Fase VII	Pruebas y creación de aplicación de prueba	1 mes
Fase VIII	Memoria	3 meses

Cuadro 7.1: Duración asociada a cada fase del proyecto

7.2. Problemas

Los primeros problemas resultaron al ver que APML era un estándar poco desarrollado y en desuso, por lo cual hubo que plantarse otras opciones.

Otro problema fue la búsqueda de un servidor OpenID que se adaptara a los requisitos, ya que existe gran variedad de servidores, los cuales tienen distintas características y están programados en varios lenguajes.

Luego fueron surgiendo distintos problemas al programar la aplicación e intentar conectar con la parte servidora. El más importante fue que no se conseguía subir certificados y gracias a que una versión anterior de WSO2 la 1.5 no utilizaba https , usaba http, se pudo analizar el tráfico de las cabeceras ip cuando se subía un certificado y así poder detectar donde se encontraba la diferencia con los datos que nuestra biblioteca enviaba.

Capítulo 8

Conclusiones y trabajos futuros

8.1. Conclusiones

Una vez finalizado el proyecto, se puede concluir que se han cumplido todos los objetivos que se marcaron meses atrás, con distintas tecnologías, de las que primeramente se había pensado, pero con grandes resultados. Al final se ha diseñado una biblioteca, junto con una aplicación que cumple con los requisitos básicos que un usuario puede requerir de esta aplicación y se ha podido demostrar con la aplicación gráfica como esta biblioteca tiene la funcionalidad deseada y es fácil de utilizar en otros desarrollos como de utilizarla a bajo nivel.

Además es una biblioteca ligera y portable a otros entornos y dispositivos como puede ser desarrollo en J2ME para dispositivos móviles.

De gran importancia es la aportación que pueden llegar a ser los apéndices realizados con los cuales se facilita el uso del servidor OpenID, en ellos se detalla tanto la instalación del servidor como la configuración inicial, la creación de certificados SSL propios, manuales usuario y administrador así como configuración de almacenes de usuarios.

Uno de los objetivos iniciales del proyecto era el aseguramiento de seguridad y utilidad del software desarrollado. Para ello, el sistema ha sido sometido a unas pruebas definidas para este fin. Los resultados han sido satisfactorios y por ello se puede concluir que el software cumple con las funcionalidades básicas para las que fue diseñado.

En conclusión, pienso que OpenID es y seguirá siendo en los próximos años, una de las mejores alternativas a la hora de ser autentificado y registrado de forma segura en cualquier entorno que lo soporte.

Esta biblioteca ha sido pensada para llegar a grandes masas de usuarios y ser accesibles desde una amplia variedad de soportes, plataformas, dispositivos, etc sin importar el lugar en el que te encuentres. Por este motivo creó que hay que seguir formándose y actualizándose y que puede llegar a ser una opción bastante importante.

8.2. Trabajos futuros

Actualmente el proyecto es una biblioteca, por lo cual deja abierto el entorno del desarrollo futuro, para el uso de nuevas aplicaciones, en distintos entornos, las aplicaciones pueden ser muy diversas como la interacción con perfiles de redes sociales, que puedan ser actualizados al momento con solo actualizar algún perfil ya existente en otra aplicación.

Otro uso podría ser la integración en aplicaciones profesionales que contengan información del usuario y poder actualizar el contenido al instante como puede ser el caso de un currículum o la vida laboral de un individuo para no tener que cambiar el contenido en varios lugares a la vez.

Existe una infinidad de posibilidades de integración de la biblioteca con distintas aplicaciones como pueden ser la agregación de fotos utilizando Flickr, introducción de listas de reproducción mediante Spotify, gustos en cuanto a películas, música, libros..., para que puedan resultar más fáciles las búsquedas, así como la agregación bookmarks para poder cargarlos desde cualquier navegador.

También sería posible la integración con otros almacenes de usuarios como pueden ser servidores LDAP que nos proveen información acerca de los usuarios registrados.

El proyecto está enfocado en un desarrollo para entornos móviles con tecnología JAVA, pero además en un futuro, también se podría hacer la integración para otros entornos móviles que utilicen Symbian o J2EE, así como otro lenguaje de programación.

También seria posible igual que interconectar varias aplicaciones con un servidor, intercambiar información entre servidores OpenID para poder tener la opción de poder tener una copia de seguridad.

Apéndice A

Instalación del servidor WSO2

WSO2 se encuentra desarrollado bajo java 1.5/1.6 por lo tanto es portable tanto en Windows como en Linux. Aproximadamente requiere unos 256MB de espacio libre en disco duro.

Para la instalación en un entorno Linux, (es similar en entornos windows) es necesario descargar previamente el paquete con el binario de la instalación que podemos encontrar en el siguiente link:

http://wso2.org/downloads/identity.

Una vez descargado, el siguiente paso es descomprimir el fichero zip.

A continuación hay que definir la variable JAVA_HOME para definir el path de la maquina virtual de java. Para que este path este siempre activo y no se tenga que introducir cada vez que ejecutemos el servidor, se puede modificar el fichero [WSO2_IS_HOME]/bin/wso2server.sh dentro del directorio de la instalación y allí al principio del fichero añadir el camino hacia la maquina virtual entre corchetes como por ejemplo:

JAVA_HOME="/usr/lib/jvm/java–6–sun/jre"
export JAVA_HOME

Ahora ya se podrá ejecutar el servidor, lo que hay que hacer es dar permisos de ejecución al [WSO2_IS_HOME]/bin/wso2server.sh y a continuación ejecutar el script.
El servidor ya estará corriendo en la siguiente URL https://localhost:9443/carbon donde se puede comprobar que esta funcionando correctamente.

Apéndice B

Configuración inicial del servidor WSO2

Lo primero que hay que configurar es el host y el puerto por el que se quiere que sea visible el servidor, para ello tenemos que modificar los siguientes ficheros:

[WSO2_IS_HOME]/conf/carbon.XML

Se modifican las siguientes lineas cambiando localhost por el host del servidor:

<ServerURL>https://localhost:${carbon.https.port}${carbon.context}/
services/</ServerURL>

<HostName>localhost</HostName>

En [WSO2_IS_HOME]/conf/mgt–transports.XML se configuran los puertos para http y https.

<transport name="http"class="org.wso2.carbon.server.transports.http.
HttpTransport"><parameter name="port">9763</parameter></tran
sport><transport name="https"class="org.wso2.carbon.server.transpor
ts.http.HttpsTransport"><parameter name="port">9443</parameter>
</transport>

Ya solo falta modificar la URL con la cual los usuarios accederán al servicio OpenID:

En [WSO2_IS_HOME]/conf/identity.XML

hay que modificar el host y el puerto que se ha modificado anteriormente.

<OpenIDServerUrl>https://localhost:9443/openidserver</OpenIDServerUrl>
<OpenIDUserPattern>https://localhost:9443/openid/</OpenIDUserPattern>

A continuación al lanzar de nuevo el servidor se puede acceder por el host y el puerto elegido y los usuarios podrán autentificarse mediante la nueva URL de la siguiente forma:

https://localhost:9443/openid/nombre_usuario

Donde se tienen que cambiar el host y el puerto por el que se ha decidido anteriormente.

Apéndice C

Creación de un certificado SSL propio

WSO2 viene con un certificado SSL [21] predeterminado. Este certificado por defecto no es adecuado para su uso en un entorno de producción, puesto que cualquiera que tenga una copia de la distribución de WSO2 también tendrá una clave privada del certificado de forma predeterminada.

La solución es generar ya sea un nuevo certificado auto firmado o, preferiblemente, instalar un certificado firmado por una entidad certificadora de confianza.

Para crear el almacén de claves realizaremos la siguiente operación en consola:

keytool –genkey –alias wso2carbon –keyalg RSA –keystore wso2carbon.jks –storepass wso2carbon

A continuación hay que responder a las preguntas que se realizan sobre nuestra organización, por ejemplo:

¿Cuáles son su nombre y su apellido?
[Unknown]: dominio
¿Cuál es el nombre de su unidad de organización?
[Unknown]: wso2carbon
¿Cuál es el nombre de su organización?
[Unknown]: wso2carbon
¿Cuál es el nombre de su ciudad o localidad?
[Unknown]: Madrid

¿Cuál es el nombre de su estado o provincia?
[Unknown]: Madrid
¿Cuál es el código de país de dos letras de la unidad?
[Unknown]: ES
¿Es correcto CN=dominio, OU=wso2carbon, O=wso2carbon, L=Madrid, ST=Madrid, C=ES?
[no]: Si
Escriba la contraseña clave para <wso2carbon>
(INTRO si es la misma contraseña que la del almacén de claves):
Volver a escribir la contraseña nueva:
wso2carbon

Esto debería crear un archivo llamado wso2carbon.jks que será nuestro contenedor de claves.

Lo siguiente es la creación certificado de solicitud de firma para introducirlo en el almacén de claves, para ello simplemente ejecutaremos el siguiente comando:

keytool –certreq –alias wso2carbon –keyalg RSA –keystore wso2carbon.jks –storepass wso2carbon

—–BEGIN NEW CERTIFICATE REQUEST—–

MIIBtTCCAR4CAQAwdTELMAkGA1UEBhMCTEsxEDAOBgNVBAg
TB1dlc3Rlcm4xEDAOBgNVBAcTB0NvbG9tYm8xEjAQBgNVBAoTCVdT
TzIgSW5jLjENMAsGA1UECxMEV1NBUzEfMB0GA1UEAxMWd3NvMi5v

(more encoded data).....

r2KjVW7Oo2ENuaL3g+Zej09v0kb0ic09oSQaUowtdHtfL2Wp
OuqwTqi81ysg9ev2rxrRX1Rp8fa5fTCaUThNDa5h3mi6fe5brGT0BLPd2eEa
BBrlDtTCWgE030bdR3zLNFYR

—–END NEW CERTIFICATE REQUEST—–

Una vez realizada esta operación, simplemente hay que cambiar el contenedor de claves que tenía el servidor por el nuevo que se ha generado.
Para ello se sobre escribirá el fichero generado wso2carbon.jks por el fichero ya existente en
[WSO2_IS_HOME]/resources/security/wso2carbon.jks.

Apéndice D

Manual de usuario WSO2

Se puede registrarse como nuevo usuario con una tarjeta de información de auto–emitidos o con contraseña de usuario basada en nombre de la cuenta.

Se puede acceder al WSO2 Identity Server con una tarjeta de la información registrada auto–emitidos o con contraseña de usuario basada en nombre de la cuenta.

En el menú lateral izquierdo se pueden visualizar My Identity que es donde se encuentran todos los datos de los usuarios con sus correspondientes configuraciones. Estos son los menús que se muestran:

- My Profiles: Los usuarios pueden crear nuevos perfiles distintos y además pueden modificar los perfiles existentes. Los campos de los perfiles pueden ser añadidos solo por el administrador.

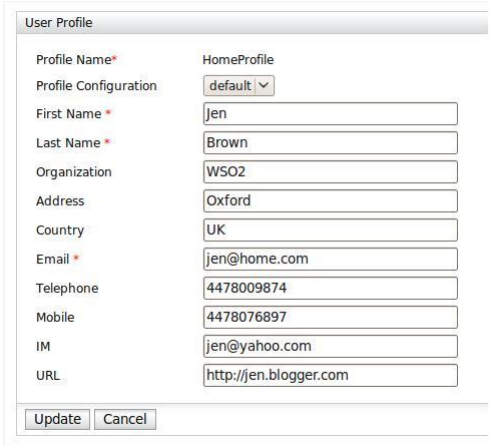

Figura D.1: Edición del perfil de usuario

- InfoCard OpenID: Permite a los usuarios descargar sus propias tarjetas de información y tarjetas de información OpenID con los datos de sus perfiles.

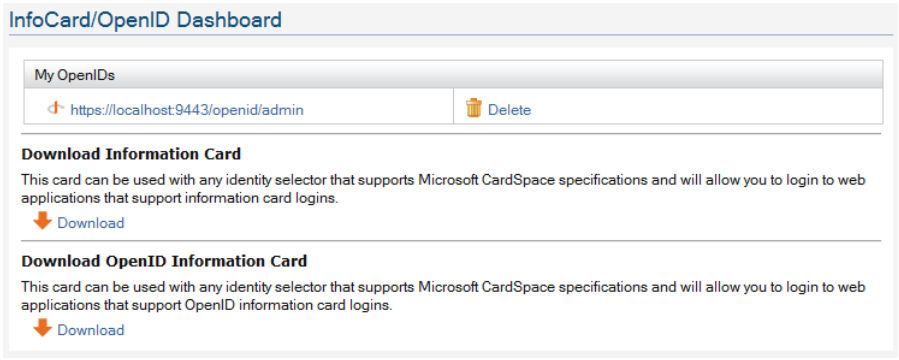

Figura D.2: Tarjetas de información personal

- Relying Parties: Se pueden subir los certificados públicos de confianza de las partes que confían.

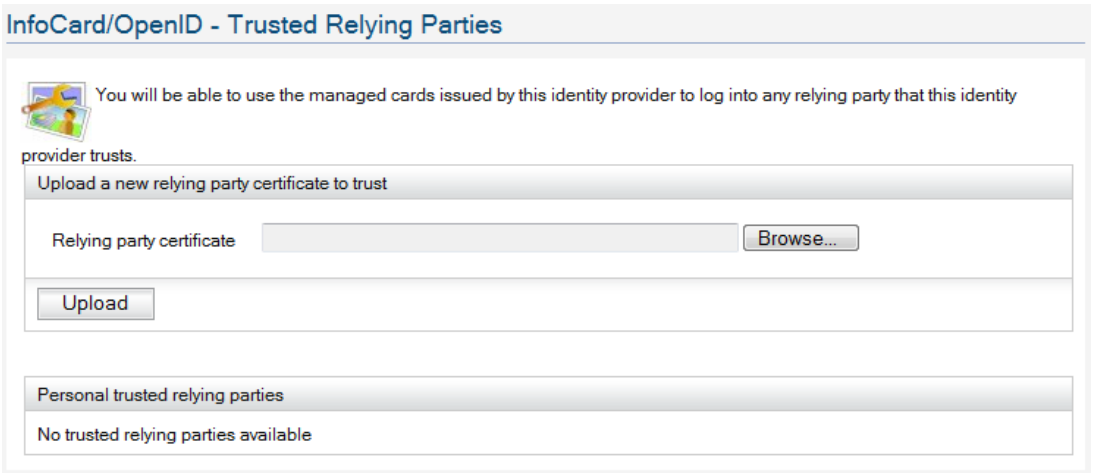

Figura D.3: Relying Parties

- Multifactor Authentication: Se puede configurar para habilitar la autenticación basada en XMPP, en la cual cuando se va loguear en el servidor, se pedirá un PIN de usuario al proveedor XMPP para aumentar la seguridad de la autenticación.

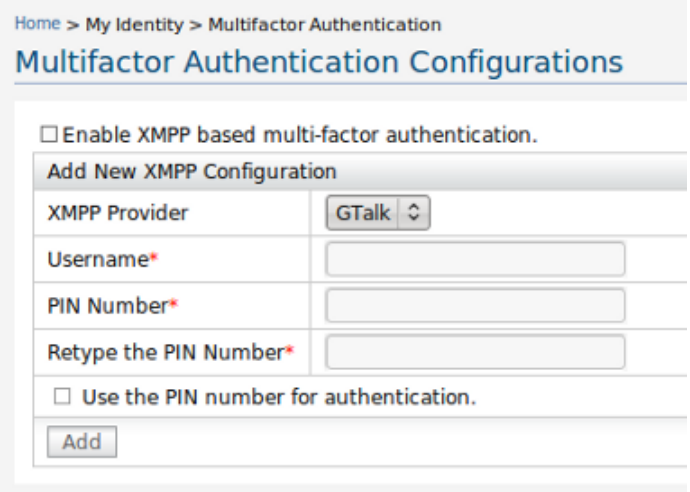

Figura D.4: Configuración de XMPP

Para realizar la activación del servicio solo se tiene que introducir los datos que pide:

Primeramente se selecciona el proveedor de XMPP que se prefiera.

A continuación especificar el nombre de usuario correspondiente al proveedor de XMPP antes mencionado. Tenga en cuenta que se debe introducir el nombre del usuario junto con la extensión del proveedor.

Ej: wso2user@xmpp–provider.com

Introduzca un código PIN que contiene más de seis caracteres, vuelva a insertar el mismo número de PIN.

La casilla de verificación, se utiliza para especificar si el número PIN debe ser utilizado en el proceso de autenticación. Si está marcada, entonces el número PIN debe ser proporcionado en el proceso de autenticación, de lo contrario sí / no es el tipo de respuesta que se espera en el proceso de autenticación.

Por último, haga clic en el botón add.

Apéndice E

Servidor OpenID WSO2, guía del administrador

Una vez instalado se puede acceder a través de la dirección WEB y el puerto que ya se ha definido. Se puede ingresar como administrador utilizando el usuario “admin” y la contraseña “admin”, dentro del panel de configuración se pueden apreciar distintas categorías:

1. Entitlement:

   Policies: Donde se describen las políticas, las cuales están definidas en XACML 2.0 y éstas pueden ser añadidas, importadas de otras existentes o evaluadas.

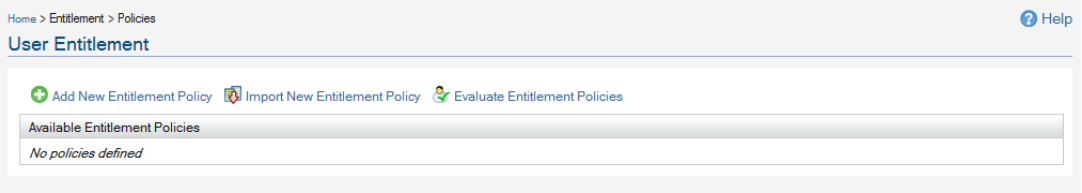

Figura E.1: Entitlement Management

2. Configure:

   User Management: Gestión de las cuentas de usuario y funciones de usuario a diferentes niveles desde donde se pueden añadir, modificar o eliminar cuentas y funciones de usuarios.

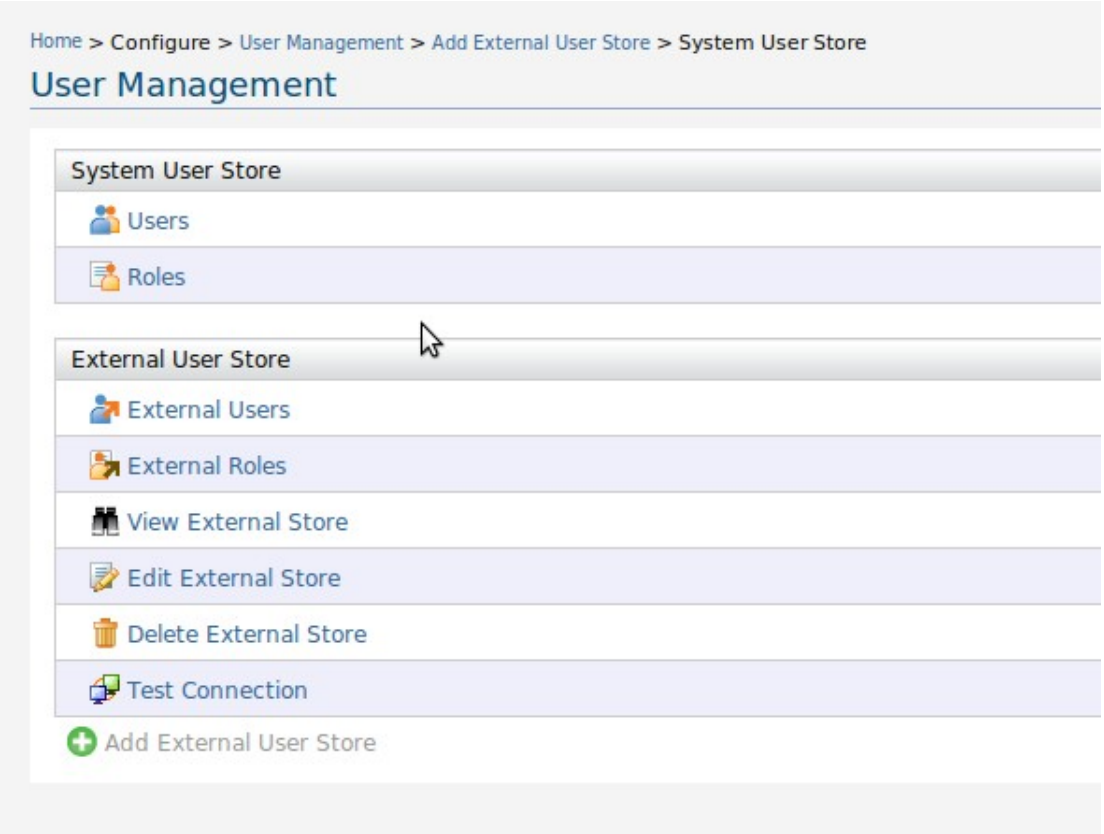

Figura E.2: User Management

Claim Management: Permite gestionar y asignar una serie de atributos a los perfiles de los usuarios. Donde se pueden definir esquemas para que distintas aplicaciones sincronicen está información. Estos esquemas indican la estructura de los datos, indican la descripción de los datos así como su orden, estructura sintáctica, si son requeridos o soportados por defecto.

Las Claim que trae por defecto son las siguientes:

http://wso2.org/claims: dialecto predeterminado para WSO2 de carbono.

http://schemas.xmlsoap.org/ws/2005/05/identity: dialecto predeterminado para las tarjetas de información.

http://axschema.org: dialecto predeterminado para OpenID atributo EXchange.

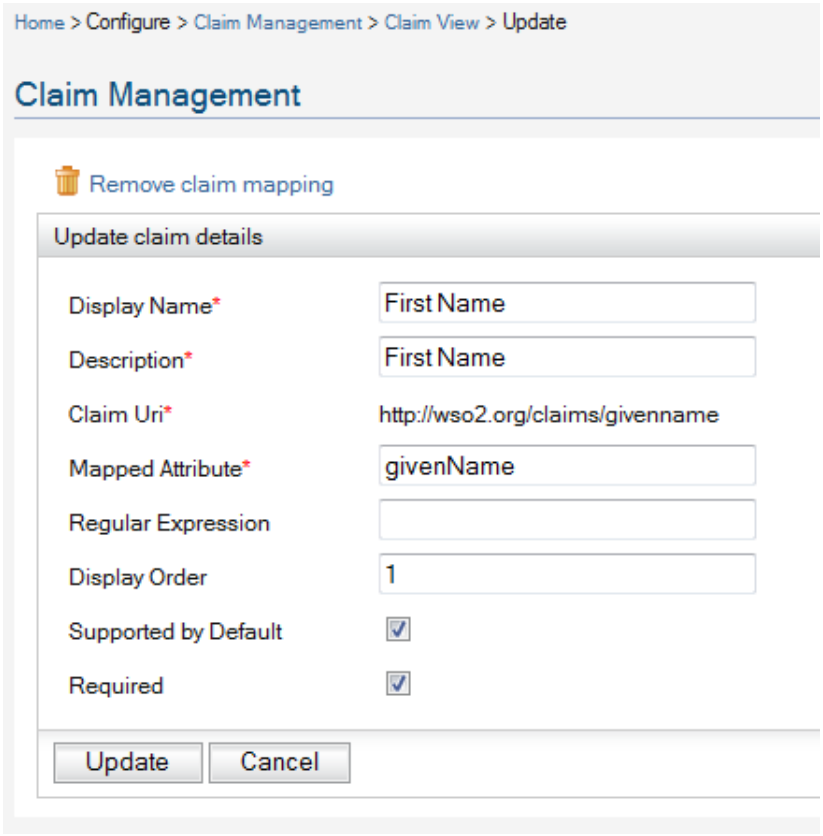

Figura E.3: Claim View

hhttp://schema.openid.net/2007/05/claims: dialecto predeterminado de Simple Registro OpenID .

Profile Management: Permite añadir / modificar y borrar la configuración del perfil. Se basa en los controles de configuración de cómo los valores vacíos de los usuarios se comportan en su perfil.

Si a una petición en el perfil de un usuario no se le da un valor explícitamente, entonces se le da un valor basado en la siguiente configuración:

- Se hereda: Esto significa recuperar el valor de la correspondiente reclamación en el perfil predeterminado y utilizarlo. Este es el comportamiento predeterminado.
- Reemplazados: Deje el valor vacío como es, no intente recuperarla en cualquier otro lugar.

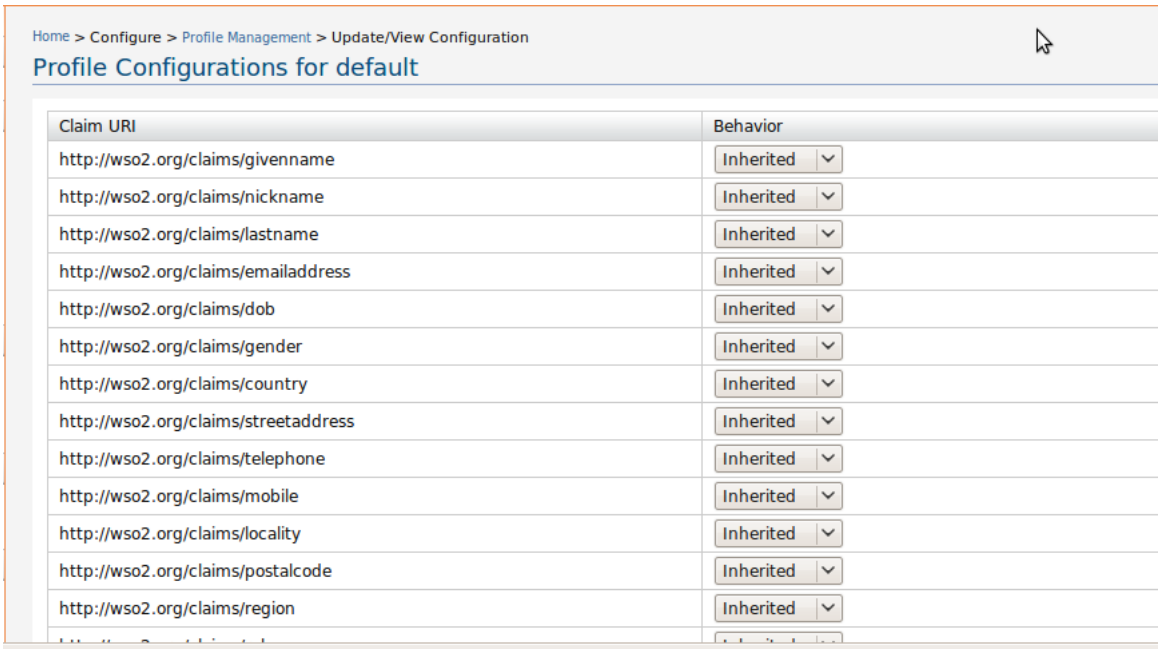

Figura E.4: Edición de Perfil

- Oculto: Ocultar está afirmación en el perfil.

Key Stores: Gestiona las claves que se almacenan en una base de datos. Un almacén de claves debe contener un par de claves con un certificado firmado por una autoridad de certificación de confianza (CA). El WSO2 utiliza el tipo de clave privada JKS llamada WSO2 de carbono. En el cual se pueden importar certificados.

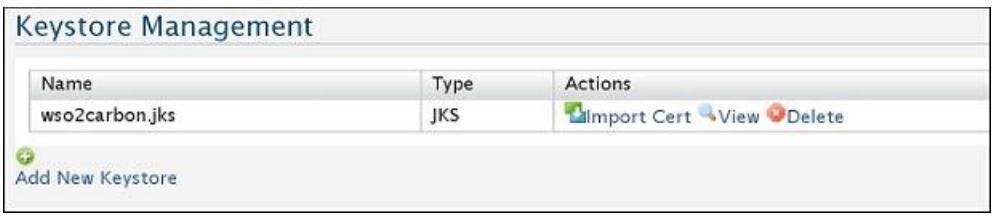

Figura E.5: Gestor de almacenamiento de claves

XKMS: WSO2 de carbono se suministra con un servicio web en construcción XKMS la confianza que se está construyendo en la parte

superior de la especificación XKMS y la cual consta de 6 servicios que pueden ser usados para simplificar la gestión de claves:

- Servicio de Registro
- Localización de servicios
- Validar el servicio
- Revocar el servicio
- Recuperar el servicio
- Reedición de servicios

3. Logging:

   Muestra la actual configuración de Log4j. Y también permite modificar la configuración existente. Se puede modificar la configuración global de Log4j. Si se selecciona la casilla de verificación se guardaran todos los cambios de configuraciones, todas las modificaciones se mantendrán y se estarán disponibles incluso después de reiniciar el servidor.

4. Manage:

   - Card Issuer:El componente de las tarjetas de información emisor del servidor de identidad permite configurar las tarjetas de configuración emitidas. Opciones:
     - Card Name: Muestra el nombre de la información de la tarjeta descargada.
     - Valid Period: El período válido de una tarjeta emitida en número de días.
     - Supporting Token Types: Tipos de token de seguridad soportados.
     - Symmetric binding used: Especifica si debe utilizarse o no vinculante simétrica.

   - Security Token Service: Para configurar la STS genéricos para emitir tokens de seguridad basados en claims.

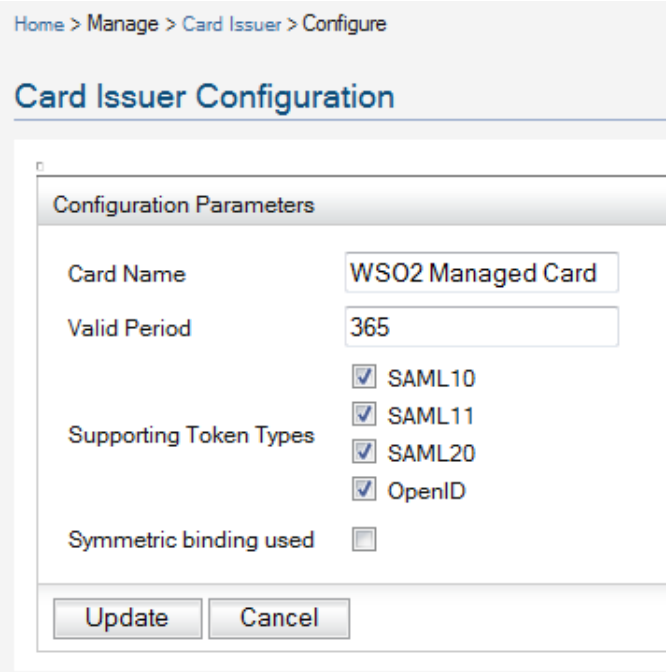

Figura E.6: Card Issuer

- Shutdown/Restart: Para realizar el apagado / reinicio del servidor. La máquina puede ser apagada normalmente o por la fuerza. Las opciones disponibles son:

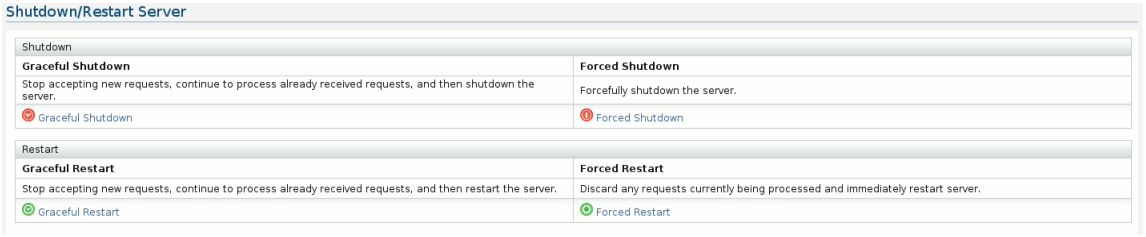

Figura E.7: Shutdown / Restart

- Graceful shutdown: Apagado normal.
- Forced shutdown: Cierre forzoso.
- Graceful Restart: Reiniciar normal.
- Immediate Restart: Reinicio inmediato.

5. Registry:

    - Browse: Este componente se puede utilizar para navegar por los recursos almacenados en el Registro.
    - Search: Todos los recursos que se encuentran en el Registro, se puede buscar a través de está interfaz.

6. Monitor:

    - System Statistics: Muestra algunas estadísticas relacionadas con la instancia de los servicios de datos WSO2. Estos incluyen memoria libre, el recuento de peticiones, el nombre del servidor, tiempo de inicio del servidor, sistema en el tiempo, los servicios activos, la memoria total, el tiempo medio de respuesta, tiempo de respuesta mínimo y tiempo de respuesta máximo.
    - System Logs: Aquí se muestran todos los registros del sistema. También se puede buscar un log en particular a través de la función Search logs.

Apéndice F

Configuración de almacenes de usuarios externos en WSO2

El servidor WSO2 [5] permite la posibilidad de conectar con almacenes de usuarios externo, y con ello el uso que los usuarios existentes y los roles.

El Administrador de usuarios soporta los siguientes formatos de tiendas de usuarios externos.

- Lightweight Directory Access Protocol (LDAP).
- Active Directory (AD).
- Base de datos relacional de usuarios personalizada.

En este caso concreto se va a centrar en la explicación de la configuración de un servidor LDAP, en el cual se pueden reutilizar los usuarios, contraseñas y perfiles de los distintos usuarios que se encuentren en el servidor LDAP, para realizar este proceso se accede como administrador al servidor WSO2 y en Home >Configure >User Management >Add External User Store y se selecciona LDAP.

Se introducen los datos de acceso al servidor LDAP, como por ejemplo para conectar con el servidor LDAP de la UC3M tenemos que rellenar los siguientes campos:

Connection URL: ldap://ldap.uc3m.es:389

Connection user name: uid=100056631,ou=CURSO DE ADAPTACION AL GRADO ING SISTEMAS DE COMUNICACIONES,ou=Alumnos,ou=Gente,o=Universidad Carlos III,c=es

Connection password: ******

User Context Name: ou=Gente,o=Universidad Carlos III,c=es

User Pattern: uid=100056631,ou=CURSO DE ADAPTACION AL GRADO ING SISTEMAS DE COMUNICACIONES,ou=Alumnos,ou=Gente,o=Universidad Carlos III,c=es

Una vez completos los campos se podrán acceder con los usuarios y contraseñas que se tienen en el servidor LDAP y los cuales ahora dispondrán de OpenID.

El administrador podrá asignar roles a los distintos usuarios así como crear distintas Claims para poder utilizar los datos que tienen los usuarios en el servidor LDAP.

Bibliografía

[1] Api java.security. http://download.oracle.com/javase/1.4.2/docs/api/java/security/package-summary.html. [Online; accedido el 16-Marzo-2010].

[2] Api javax.net.ssl. http://download.oracle.com/javase/1.4.2/docs/api/javax/net/ssl/package-summary.html. [Online; accedido el 16-Marzo-2010].

[3] Apml. http://apml.areyoupayingattention.com/. [Online; accedido el 13-Noviembre-2009].

[4] Clamshell. http://wiki.guruj.net/Clamshell!Home. [Online; accedido el 26-Noviembre-2009].

[5] Facilelogin. http://blog.facilelogin.com. [Online; accedido el 19-Enero-2010].

[6] Instalar openid en tu site. http://g05l21.net/2008/07/04/instalar-openid-en-tu-site/comment-page-1/. [Online; accedido el 18-Noviembre-2009].

[7] Openid. http://openid.net/. [Online; accedido el 18-Noviembre-2009].

[8] phpmyid. http://www.siege.org/projects/phpMyID/. [Online; accedido el 26-Noviembre-2009].

[9] Wso2. http://wso2.com/products/identity-server/. [Online; accedido el 8-Diciembre-2009].

[10] Estableciendo conexiones http a moodle desde j2me. http://33pfc2006.blogspot.com/2006/11/pues-eso-lo-prometido-es-deuda.html, 2006. [Online; accedido el 16-Marzo-2010].

[11] G. Álvarez. ¿cómo instalar ssl en un servidor? http://www.iec.csic.es/CRIPTONOMICON/consejos/instalarssl.html, 1997. [Online; accedido el 5-Noviembre-2009].

[12] Drummond Reed David Recordon. Openid 2.0: a platform for user-centric identity management. http://doi.acm.org/10.1145/1179529.1179532. Actas del segundo congreso del ACM de Digital identity management, páginas 11-16, ISBN 1-59593-547-9, ACM, NY USA, 2006.

[13] B. Eckel. *Piensa en Java*. Pearson Educación SA, 2a ed. Madrid., 2002.

[14] R. Fernando. *Setting Up Keystores for a Client and a Service*. 2006.

[15] J García. Aprenda java como si estuviera en primero. http://www.tecnun.es/asignaturas/Informat1/ayudainf/aprendainf/Java/Java2.pdf, 2000. [Online; accedido el 3-Febrero-2010].

[16] Robin Good. Apml o perfil de atención: Guía para principiantes. http://www.masternewmedia.org/es/2008/01/31/apml_o_perfil_de_atencion.htm. [Online; accedido el 13-Noviembre-2009].

[17] Franz Inc. Ssl tutorial. http://www.franz.com/support/tutorials/ssl-tutorial.htm. [Online; accedido el 5-Noviembre-2009].

[18] D. Maroto. *PFC: Estudio sobre seguridad en dispositivos móviles con sistema operativo Symbian*. 2008.

[19] Verisign. Ssl manual. http://tvsecure.net/manual/manual3.html. [Online; accedido el 5-Noviembre-2009].

[20] OpenID Wiki. Run your own identity server. http://wiki.openid.net/Run-your-own-identity-server. [Online; accedido el 27-Noviembre-2009].

[21] WSO2. Setting up keystores for a client and a service. http://wso2.org/library/174. [Online; accedido el 5-Noviembre-2009].